VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Porovnání softwarových procesů UP, Scrum, XP

Comparison of Software Processes UP, Scrum, XP

2010 Jana Kaděrová

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 7. května 2010 .....

Ráda bych poděkovala svému vedoucímu, panu Ing. Svatopluku Štolfovi Ph.D., za cenné rady a čas, který mi věnoval, protože bez jeho odborného vedení by tato práce nevznikla.

Abstrakt

Bakalářská práce se zabývá třemi softwarovými procesy: Unifikovaným procesem vývoje softwaru, procesem Scrum a Extrémním programováním. Jejím obsahem je jak obecná teorie ze softwarového inženýrství, tak i teorie a popis činností ve výše zmíněných metodikách. Dále se zaměřuje na samotné porovnání jednotlivých částí softwarových procesů na základě jejich vstupů a výstupů. Poslední část práce je tvořena společným slovníkem pojmů.

Klíčová slova: Softwarové inženýrství, Softwarový proces, Unifikovaný proces vývoje softwaru, Vývojový proces Scrum, Extrémní programování

Abstract

This Bachelor's thesis is about three software processes: Unified Software Development Process, Scrum Development Process and Extreme programming. Its contents is general theory of software engineering and also theory and process description of above mentioned methodologies. It also focuses on comparsion of each part of software processes based on their input and output itself. Last part of thesis is formed by common terms dictionary.

Keywords: Software Engineering, Software Process, Unified Software Development Process, Scrum Development Process, Extreme Programming

Seznam použitých zkratek a symbolů

RUP	–	Rational Unified Process
Scrum	–	Scrum Development Process
UP	–	Unified Software Development Process, Unifikovaný proces vývoje softwaru
XP	–	Extreme Programming, Extrémní programování

Obsah

Seznam obrázků

1 Úvod

Cílem této bakalářské práce je přehledně popsat metodiky Unified Software Development Process, Scrum Development Process a Extreme Programming a následně provést jejich srovnání na základě rolí, artefaktů a činnosti, které vstupují do procesu a výstupních činností, a na závěr sestavit společný slovník pojmů pro výše zmíněné metodiky.

V první částí práce je nezbytné si osvětlit základní teorii a terminologii týkající se softwarového inženýrství, používanou také v dalších částech bakalářské práce. Jsou zde rozděleny a srovnány jednotlivé softwarové procesy a stručně představeny zadané metodiky. Ve třetí, čtvrté a páté kapitole této práce jsou postupně představeny jednotlivé metodiky, společně s časovým dělením, popisem základních pracovních postupů, činností a rolí zastoupených v týmu při vývoji softwarových produktů.

Následující část bakalářské práce se zabývá samotným srovnáním vybraných částí metodik na základě vstupů a výstupů, které se v jejich průběhu vyskytují. Je zde chronologicky popsán průběh vývoje v metodikách Unified Software Development Process, Scrum Development Process a Extreme Programming.

Poslední část práce tvoří společný slovník klíčových pojmů metodik Unified Software Development Process, Scrum Development Process a Extreme Programming, které se v průběhu jejich použití vyskytují.

Závěrem jsou shrnuty výsledky práce a dále jsou zde uvedeny náměty k dalšímu výzkumu.

2 Úvod do problematiky

2.1 Softwarové inženýrství

Softwarové inženýrství má mnoho definic. Podle Fritze Bauera je softwarové inženýrství „zavedení a požívání řádných inženýrských principů tak, aby bylo dosaženo ekonomické tvorby softwaru, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“ [2] V této definici je popsáno vše, co softwarové inženýrství obsahuje, čím se zabývá a o co se snaží.

Jiná definice, podle IEEE (The Institute of Electrical and Electronics Engineers), říká: „Softwarové inženýrství je aplikace systematických, disciplinovaných a měřitelných postupů na vývoj, provoz a údržbu software, tedy použití obecných inženýrských principů na software. “ [7]

2.1.1 Softwarová krize

Softwarové inženýrství vzniklo jako samostatný a uznávaný obor díky Softwarové krizi v šedesátých letech 20. století. Charakteristickými znaky bylo neúnosné prodlužování a prodražování vývoje softwarových produktů, jejich nízká kvalita a velice špatná, či dokonce nemožná údržba a inovace, která byla potřebná pro přizpůsobení se měnícím se požadavkům zákazníků, dále také neefektivita vývoje softwaru a mnoho dalších. Mezi důvody, proč došlo k této krizi je hned několik.

Jeden ze zásadních důvodů je špatná komunikace a to při všech stádiích vývoje softwarového produktu, tedy v komunikaci zákazníka s analytikem (v té době to byl většinou přímo programátor, který mimo jiné obstarával jakousi analýzu projektu), dále komunikace tohoto analytika s vývojářem, komunikace mezi vývojáři a v neposlední řadě taky komunikace vývojářů s vedoucím projektu. V dnešní době je funkce analytika ceněna v mnohých případech více než funkce vývojáře, což je dáno hlavně tím, že nedostatečná komunikace mezi zákazníkem a týmem vyvíjející softwarový produkt by vedlo k nesplnění požadavků na něj kladených.

Nesprávný přístup vývojářů vedlo k nespokojenosti zákazníků, kteří byli přesvědčování o tom, že „jedině programátoři mají pravdu, protože pouze oni vědí, jak jde daný produkt či problém naprogramovat“. V dnešní době jsou používány například systémy CRM (Customer Relatioship Management, systém řízení vztahů se zákazníky), které vedou k eliminaci těchto problémů a ke zlepšení komunikace se zákazníkem.

O softwarovou krizi se také zasloužily nesprávné odhady, a to ať už odhady času, ceny, efektivity, tak i rozsahu celého projektu. Důraz byl kladen hlavně na psaní kódu, tím se zanedbávala analýza celého projektu a z toho zase plynulo podceňování, nebo dokonce ani neprovádění, testů beta verzí a verzování. Občas se také neprovádělo zálohování již napsaného kódu, což ovlivňovalo do značné míry efektivitu práce jednotlivých vývojářů. V dnešní době vyplynulo z analýz, že firmy, které kladou důraz na kvalitu svých produktů, mají průměrný denní nárust 30 až 50 řádek kódu programu u každého svého programátora. I přes veškerou snahu o vytvoření optimálního odhadu času často

bývají problémy, jelikož se vytváří na samém začátku vývoje produktu, kdy je k němu nejméně dostupných informací.

S předchozím problémem souvisí i otázka špatného plánování. Pro tehdejší vývojový tým bylo velmi obtížné vypracovat takový plán projektu, aby vyhovoval jak zákazníkovi, tak aby byl splnitelný i pro vývojáře. Proto se většinou upouštělo od zpracování kvalitního plánu a spoléhalo se na to, že „se to stihne“. V současnosti máme možnost využít již zpracovaných postupů a metod pro plánování jednotlivých fází projektu, jako je například plánování nákladů, časové plány, plány na využívání různých zdrojů a podobně. To vše nám však nepomůže odbourat tento problém úplně, a je doporučeno provádět několikrát během celého vývoje softwarového produktu průběžné a iterativní plánování společně s opakovanou revizi, zda plán odpovídá realitě a naopak. V minulosti vývoj jako takový probíhal tak, že bezprostředně po zadání požadavků od zákazníka programátoři zasedli k počítačům a bezhlavě psali množství kódu. Mnozí se k této metodě po letech nelibosti opět vrací a tvrdí, že je takto zefektivněn přístup k vývoji. Rozdílem je však provázanost s analýzou, návrhem a hlavně testováním. Jedním ze zástupců takové metodiky je například Extrémní programování.

Jednou z dalších potíží byla i nízká produktivita práce programátorů. Bylo to způsobeno především tím, že nevěděli, kterou část softwarového produktu mají přesně naprogramovat. Proto se snažili sepsat co největší množství kódu, aby přibylo co nejvíce funkčnosti bez ohledu na to, jaký programový kód by bylo vhodné psát nejdříve a aby se výsledné části produktu mohly provázat. V mnohých případech byli takový programátoři dokonce i úspěšní. Ve většině případů se špatné či dokonce chybějící řízení podepsalo na tom, že velké množství kódu muselo být zahozeno nebo přepracováno a vývoj byl díky tomu drahý a hlavně trval dlouho. V současnosti je trendem snaha o zastoupení různých rolí v týmu a to tak, že každý vykonává to, v čem je nejsilnější a tím zvýší efektivitu práce týmu. Obecně je nyní kladen mnohem větší důraz na koordinaci vývoje, na to, aby jednotliví členové byli „vedeni správným směrem“ a aby jednotlivé části programového kódu spolupracovaly a dokázaly spolu komunikovat.

Další příčinou problémů při vývoji byla i neznalost základních pravidel, což mohlo být způsobeno například chybějícími zkušenostmi. Například Brooksův zákon z roku 1975 říká:„Přidání řešitelské kapacity u zpožděného softwarového projektu způsobí jeho další zpoždění“. O tomto a mnoha dalších pravidlech v době softwarové krizi málokdo věděl a jejich platnost a existenci si vývojáři uvědomili s přibývajícími zkušenostmi až v průběhu několika let. Dnešní vývojáři si platnost těchto základních pravidel na plno uvědomují a řídí se jimi.

Předposledním problémem bylo podceňování hrozeb a rizik, které se v různých fázích vývoje vyskytly. Mnohé z nich mohly být bez problému a s minimálními náklady vyřešeny ihned při jejich prvotním výskytu, nicméně většina členů vývojového týmu jim nepřikládala význam a důležitost. V současnosti je naopak snaha o odhalení chyb tam, kde je nečekáme a někdy ani nepředpokládáme. Řada z dnes již známých metodik předepisuje pravidelné provádění rozboru všech potenciálních hrozeb, které by se mohly objevit během práce na projektu. Tento rozbor nazýváme analýza rizik.

Nakonec by chtělo zmínit něco o problému nezvládnuté technologie, případně domněnku, že zavedením nové technologie do praxe se odstraní stávající problémy. Tato představa sice někdy přetrvává dodnes, přesto však už i samotní vývojáři začínají chápat, že na začátku dobře zvolená technologie může vést k úspěšnému konci vývoje softwarového produktu, avšak není to záruka. Je třeba brát v úvahu i ostatní potencionální problémy, které se mohou vyskytnout v průběhu vývoje.

Softwarová krize měla mnoho příčin, avšak pro softwarové inženýrství jsou důležité její důsledky. Na základě nich se totiž začaly formovat propracované postupy pokrývající všechny fáze vývoje softwaru, které by při dodržování měly odstranit charakteristické problémy krize. Musíme však brát v úvahu i to, že ani dnešní projekty nemusí skončit vždycky včas a s použitím na začátku stanovených prostředků. Je to v jisté míře způsobeno tím, že se jedná o činnost, kterou nejde spolehlivě naplánovat od začátku do konce a její náročnost a rozsah se zvyšují, také se zvyšují nároky na rychlost dodání zákazníkovi a vždy se může objevit nečekaný problém. Pečlivým výběrem a používáním vhodných metodik a postupů se může riziko selhání značně snížit.

2.2 Softwarový proces

Proces je množina částečně uspořádaných kroků vedoucí k cíli (k produkci a údržbě žádaného softwarového výsledku). Softwarový proces zahrnuje množinu souvisejících artefaktů, lidí a počítačových zdrojů, organizačních struktur a omezení.

2.3 Metodologie, metodika, metoda

Metodologie je nejobecnější pojem. Je to vědní disciplína zabývající se definicemi metodik, jejich rozborem, tvorbou, aplikací a podobně. Jedná se v podstatě o nauku o metodikách.

Metodika je komplexní návod nebo doporučení, jak postupovat při vývoji softwarového produktu a pokrývá celý jeho životní cyklus.

Metoda je označení pro konkrétní postup vedoucí k vyřešení dílčího problému při vývoji.

2.4 Rigorózní metodika

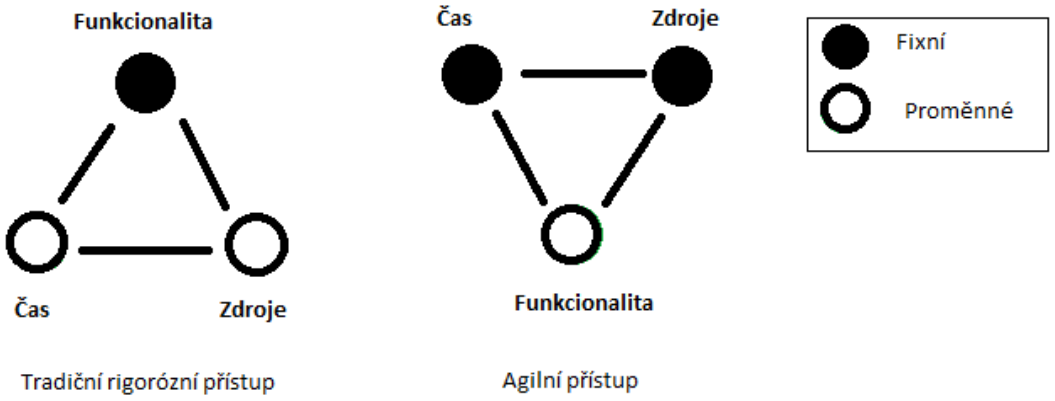
Rigorózní metodiky jsou většinou založeny na vodopádovém vývoji, to znamená že jednotlivé fáze (plánování, analýza, návrh, implementace a nasazení) probíhají za sebou. Existují ale také jiné metody, založené na iterativním a inkrementačním vývoji jako například Unified Process, kterému je věnována další část práce. Iterativní vývoj by se dal popsat jako opakované (iterativní) provádění jednotlivých fází vývoje. Výsledkem každé takovéto iterace je funkční verze, ať už výsledného produktu jako celku (rozšíření produktu) nebo jen jeho části (produkt se vyvíjí po přírůstcích). Každá iterace by neměla být příliš dlouhá, doporučují se dny, maximálně týdny.

2.5 Agilní metodika

Agilní metoda vývoje je založena na základech Iterativního vývoje softwaru, avšak je obohacena o několik aspektů. Agilní metodiky umožňují vytvořit řešení velmi rychle a pružně jej přizpůsobit měnícím se požadavkům. Hlavní myšlenkou, jak ověřit správnost navrženého produktu je vytvořit jej nebo jeho část, co nejrychleji, předat zákazníkovi k otestování a na základě zpětné vazby jej upravit. Cenou za flexibilitu a přizpůsobivost je však obvykle menší rozsah projektu, které je pomocí agilních metodik možné zvládnout. Obecně jsou agilní metodiky využívané především v menších vývojových týmech a na menší projekty. To však nemusí být nutně pravidlem.

2.6 Srovnání rigorózních a agilních metodik

Rozdíl mezi těmito metodikami bude zřejmý z obrázku 1 :



Obrázek 1: Tři proměnné ovlivňující vývoj produktu

Jak je vidět, při tradičním rigorózním vývoji softwaru jsou požadavky neměnné a jsou stanoveny na začátku vývojového procesu. Hlavním cílem je jejich splnění. Proměnné jsou zdroje a čas. Z toho vyplývá, že na začátku je jasné,co daný produkt bude umět, ale těžko se dá odhadnout, kolik bude stát a jak dlouho bude vývoj trvat.

Opakem jsou agilní metody vývoje. Při nich se na začátku stanoví čas na zrealizování a množství zdrojů, které je zadavatel ochoten akceptovat. Funkcionalita a požadavky se průběžně mění nebo přizpůsobují.

2.7 Stručné představení metodik

2.7.1 Unified Software Development Process

Metodika Unified Process (UP) je zástupcem tradiční metodiky vývoje softwaru, založeném na iterační a inkrementální tvorbě. UP je velmi rozsáhlá a propracovaná, je řízena uživatelskými požadavky a rizikem. Její efektivita při vývoji jednodušších softwarových produktů je poněkud nižší. Hlavním důvodem je dlouhý časový úsek nutný k vytvoření procesu „na míru“ a k nasazení samotné metodiky k vývoji produktu. Z toho plyne, že UP není až tak vhodné pro jednorázové úkoly.

2.7.2 Scrum Development Process

Scrum Development Process je agilní proces pro rozvržení a řízení vývoje. Je založený na týmovém iterativním a inkrementačním přístupu při vývoji systému nebo produktu, kde se často mění požadavky. Iterace, neboli Sprint trvá přibližně 30 dní.

2.7.3 Extrémní programování

Extrémní programování je opět agilní metodika vývoje, která se opírá o osvědčené metody klasického programování, jako například revize a testování kódu, integrace, krátké iterace, programování ve dvojici a další, avšak je dovedena do extrému, což vede k přizpůsobení se měnícím se požadavkům zadavatele práce a vyšší kvalitě výsledného produktu.

3 Unified Software Development Process

3.1 Úvod

Pojem Unified Software Development Process (unifikovaný proces vývoje softwaru) se běžně zkracuje jen na Unified Process (UP). Tato metodika je velice podobná komerční Rational Unified Process (RUP), avšak na rozdíl od ní je otevřeným standardem, který je možno bezplatně použít k vývoji vlastních produktů. Ten, kdo se rozhodne pro využívání UP se musí smířit s tím, že nebude mít k dispozici doplňkové nástroje, které by mohl získat při zakoupení licence k využívaní metodiky RUP. Přesto má tato metodika spoustu uživatelů, kterým pro jejich objektový, iterativní a inkrementální vývoj plně dostačuje.

Historické kořeny metodiky UP sahají až do roku 1967, kdy vznikl tzv. Ericssonův model, jehož hlavní myšlenkou bylo rozčlenění složitých systémů na množiny vzájemně propojených bloků. Ty byly propojeny tak, aby vytvářely větší stavební bloky, které poté tvořily celou konečnou podobu finálního systému. V dnešní době se takovémuto postupu vývoje říká komponentový vývoj softwaru.

Obecně bývá za otce metodiky UP považován Ivar Jacobson, který v roce 1987 založil ve Stockholmu firmu Objectory AB. Ta stojí za vznikem metodiky Obejctory založenou na Ericssonově modelu. Zde se poprvé v historii objevuje použití konceptu případů užití (use-case) a objektově orientovaného návrhu. O několik let později, v roce 1999, publikoval autor knihu s názvem Unified Software Development Process, jejíž náplní, jak už z názvu plyne, je popis metodiky UP. Na rozdíl od metodiky RUP, která je produktem firmy Rational, je UP otevřeným standardem od tvůrců jazyka UML. Metodiky RUP tedy vychází z nekomerčního Unified Process. [3]

Obě metodiky, jak UP tak RUP, jsou zástupci tradičních, tedy rigorózních metodik. Charakteristické pro obě je to, že jsou velmi robustní a propracované, využívají se především pro větší projekty a rozsáhlejší týmy. Metodika RUP je nejrozšířenější a dalo by se možná říct i nejúspěšnější komerční variantou metodiky UP. RUP je založena na stejných principech a myšlenkách jako UP, avšak v současnosti UP podstatně rozšiřuje. Rozdíly mezi nimi jsou však především v podrobnostech a v hloubce propracování. Smysl obou metodik je podobný.

3.2 Popis

Metodika je UP je úzce propojená s jazykem UML. Veškeré činnosti a modely, které se při vývoji objeví, jsou zachycovány prostřednictvím diagramů, které UML nabízí. Cílem metodiky je zodpovězení veškerých otázek ve vývojovém procesu začínající kdo, co, kdy a jak.

UP, stejně jako RUP, je jednou z obecných metodik tvorby softwarového vybavení. Autoři vnímají každý projekt jako zcela odlišný a nelze použít jednu a tutéž metodiku pro vývoj jakéhokoliv produktu jakýmkoli týmem. Z toho plyne, že pro každý projekt je nutné vytvářet novou, konkrétní instanci této metodiky.

Axiomy metodiky UP:

1. případy užití a riziko řídí vývoj
2. architektura je jedna z nejdůležitějších části při vývojovém procesu
3. vývoj probíhá iterativním a inkrementálním (přírůstkovým) způsobem

Klíčovým artefaktem při návrhu a vývoji je textový popis použití daného vyvíjeného softwarového produktu jeho konkrétním uživatelem, tedy případ použití. Na jeho základě jsou stanovovány další postupy vývoje a zachycovány požadavky, které na produkt klade zákazník.

Další neodmyslitelnou části vývoje je analýza rizik. Jelikož rizika se nachází všude kolem vývojového procesu, je nutné se na ně připravit, vyhledávat je dopředu tak, abychom mohli co nejdříve najít jejich řešení.

Metodika UP říká: software je kvalitní pouze tehdy, je-li pečlivě navržen. Z toho tedy plyne neoddělitelná část vývoje, tedy návrh. Tím vzniká zásadní rozdíl oproti agilním metodikám, kde na návrh není kladen až takový důraz jako u tradiční rigorózní UP.

Vývoj softwarového produktu metodikou UP probíhá iterativně a inkrementálně, jak už bylo řečeno. Iterace je v podstatě rozdělení vývoje většího projektu na řadu menších „podprojektů“, které lze snadněji zvládnout a k zásadním procesům, jako je analýza a návrh, se vývojový tým v průběhu vrací hned několikrát. Dodání produktu zákazníkovi probíhá průběžně vždy po skončení iterace, tedy po vyřešení „podprojektu“ a doplnění produktu o nové funkce.

3.3 Pracovní postupy iterace

Iterace vývoje obsahuje všechny tradiční prvky softwarového procesu:

- plánování
- analýzu a návrh
- implementaci
- testování a integraci
- dodání (at'už zákazníkovi, nebo jen v rámci vývojového týmu)

Každá iterace produkuje vlastní základní linii, která obsahuje jednu z částí verze finálního softwarového produktu a jeho dokumentaci. Postupem času jsou s každou iterací vrstveny základní linie tak dlouho, než je dosaženo konečné podoby finálního produktu.

Rozdíl mezi dvěma základními liniemi je označován jako inkrementace, neboli přírůstek. Odtud tedy plyne označování metodiky za iterační a inkrementální. Přírůstky jsou dílčími kroky, směřující k finálnímu odevzdání vyvíjeného softwarového produktu.

V každé iteraci existuje pět základních pracovních postupů metodiky UP, které určují co a jakým způsobem je třeba udělat. Kromě nich se nacházejí v iteraci i další pracovní

postupy, jako plánování, odhad a dále vše, co je specifické pro danou iteraci. Tyto postupy jsou však nad rámec metodiky UP a nejsou v ní zajištěny.

Pět základních pracovních postupů:

- **Stanovení požadavků** — Klíčovým při vývoji softwarového produktu je stanovení požadavků, které jsou na něj kladeny. To však ve většině případů není triviální, jelikož zákazník většinou není schopen přesně a jasně formulovat, co požaduje.
- **Analýza** — Analýza se obvykle provádí na daných požadavcích, které se strukturují a případně rozdělují do kategorií podle ohodnocení důležitosti na jejich implementaci.
- **Návrh** — V procesu návrhu je hlavním cílem správně zachytit všechny požadavky v architektuře produktu.
- **Implementace** — V tomto procesu se píši jednotlivé části programového kódu a to jak jednotlivých modulů a tříd, tak i všech funkcí.
- **Testování** — Ověřuje se, zda implementace funguje podle očekávání.

Každá iterace může obsahovat všech pět pracovních postupů, avšak na některé postupy může být kladen větší důraz a to v závislosti na tom, kde se vývoj a daný postup nachází v životním cyklu softwarového produktu.

Pružný přístup k plánování je zajištěn rozložením vývoje produktu na sled iterací, kdy dokončení jedné iterace vede k zahájení další. Iterace se mohou provádět také souběžně a často se tak děje. V takovém případě je ale nutné pečlivé plánování a dobře rozumět závislostem mezi artefakty každé iterace, aby nedocházelo ke snižování efektivity práce v důsledku čekání na dokončení některého artefaktu. Využíváním souběžných iterací lze dosáhnout zkrácení vývoje celého produktu.

3.4 Fáze

Životní cyklus vývoje softwarového produktu metodikou UP je rozdělena do čtyř fází: zahájení, rozpracování, tvorba a předání. V každé fázi je soustředěna pozornost jednoho nebo více pracovních postupů na dosažení cíle dané fáze, což je významný milník v životním cyklu každého produktu. Jednotlivé fáze jsou vždy zakončeny předem definovaným mezníkem, který může být výsledkem určitého artefaktu, jiné mezníky však nikoliv. Splnění mezníku je tedy známkou pokroku ve vývoji. Každá fáze se může skládat z jedné nebo několika iterací, což se odvíjí od velikosti projektu. V každé iteraci může proběhnout i všech pět pracovních postupů (stanovení požadavků, analýza, návrh, implementace a testování) a libovolný počet dodatečných. Délka iterací je pružná, obecně však platí, že žádná iterace metodiky UP by neměla trvat déle než dva až tři měsíce.

Při vývoji se v různých fázích mění i objem jednotlivých prací vykonávaných v každém z pěti základních pracovních postupů.

V prvotních fázích je největší úsilí a pozornost věnováno sestavování požadavků, analýze a plánování celého vyvíjeného produktu.

Ve fázi rozpracování pokračuje analýza, která je v této části vývoje nejdůležitější. Dále probíhá návrh, jak by se měla implementovat funkcionalita produktu a tvoří se architektura celého vyvíjeného softwarového produktu.

Fáze tvorby je i nadále zaměřená na návrh, ale důraz se začíná klást i na samotnou implementaci. Objevují se první provozuschopné verze.

Testování probíhá průběžne, s výjimkou fáze zahájení, kdy není obvykle co testovat. Je mu věnováno stejné množství času ve všech dalších fázích a iteracích.

1. Fáze zahájení

Cíle

Ve fázi zahájení se provádí cela řada úkonů. Jedním ze zásadních je tvorba podmínek proveditelnosti. Tato tvorba může na příklad zahrnovat technický návrh prototypů, což jsou modely předmětů reálného světa a simulace jejich činností za pomoci výpočetní techniky. Prototypy umožňují ověřit validitu aplikačních požadavků a na základě výsledků provést případnou korekturu koncepčního návrhu.

Dalším úkonem, který se provádí ve fázi zahájení, může být zachycení požadavků, které pomůžou s definicí rozsahu vyvíjeného softwarového projektu. V neposlední řadě se také provádí detekce kritických rizik, které by mohly zásadním způsobem ovlivnit další vývoj produktu.

Ve fázi zahájení mají nejdůležitější roli ve vývojovém týmu manažer projektu a systémový projektant.

Zaměření fáze zahájení

Fáze zahájení klade důraz na ty pracovní postupy, které se zabývají specifikací požadavků a jejich analýzou. Dále se můžou vykonávat i určité návrhářské či implementační pracovní postupy, především však tvorba prototypů, jenž by potvrzovaly správnost koncepce, jak už bylo zmíněno výše. Ve fázi zahájení ve většině případů nedochází k pracovnímu postupu testování, a to buď kvůli tomu, že není co testovat a nebo kvůli toho, že vytvořené prototypy bývají v dalších fázích zahozeny, takže je také není třeba testovat.

Milník

Ve fázi zahájení je milník předmět životního cyklu (Life Cycle Objectives) a rozsah systému. Aby mohl být milník považován za dosažený je nutné splnit následující podmínky:

- zadavateli projektu je dodán a následně jím schválen dokument obsahující hlavní požadavky kladené na vyvíjený produkt, jeho funkce, rozsah a všeobecné podmínky.

- je vytvořen počáteční případ užití
- manažer projektu odhadl rizika
- je načrtnuta architektura produktu, jenž se bude vyvíjet

2. Fáze rozpracování

Cíle

Fáze rozpracování si klade za hlavní cíl vytvoření spustitelného architektonického základu. Mělo by se jednat o skutečný spustitelný produkt, ne jen o prototyp, který je možné poté zahodit. Tato počáteční verze je vytvořena na základě specifikací architektury a postupně během dalších prací je rozšiřována, až se vyvine v plnohodnotný softwarový produkt tak, jak si jej zákazník představoval. K tomu však dojde až během fáze tvorby a poslední fáze předání.

Zaměření fáze rozpracování

Téměř veškerá pozornost fáze rozpracování je upřena na pracovní postupy zabývající se upřesněním rozsahu systému a požadavků, které jsou na něj kladeny, analýzu a návrh stabilní architektury. S blížícím se koncem fáze však nabývá na důležitosti implementace spustitelného základu vyvíjeného softwarového produktu.

Milník

Milníkem fáze rozpracování je architektura (Life Cycle Architecture). Na konci fáze rozpracování jsou detailně zkoumány předměty systému a rozsah, výběr vhodné architektury a výsledek analýzy základních rizik. Pokud jsou splněny následující podmínky, je možné považovat milník za dosažený. Podmínky:

- byl naimplementován spustitelný robustní a odolný architektonický základ vyvíjeného softwarového produktu, ve kterém byla vyřešena důležitá rizika
- byla provedena revize odhadu rizik
- byl vytvořen projekt do dostatečné hloubky, aby bylo možné sestavit realistické nabídky, které by zahrnovaly odhad času, financí a prostředků pro nadcházející fázi a zainteresované osoby s ním souhlasí

3. Fáze tvorby

Cíle

Cílem fáze je nejprve splnit veškeré požadavky analýzy a návrhu a následně vyvinout ze spustitelného základu, který byl vytvořen ve fázi rozpracování, konečnou podobu produktu. Klíčové je především zachování integrity architektury vytvářeného produktu. V mnohých případech se totiž stává, že se začne klást důraz především na programový kód a dojde tím k narušení původní vize, což obvykle vede ke snížení kvality výsledku a následnému nárůstu nákladů na jeho údržbu.

Zaměření fáze tvorby

Pozornost v této fázi je zaměřena na pracovní postup implementace, jelikož požadavky jsou již zachyceny, analýza je zpracována a je vytvořen i návrh, to vše v předešlé fázi.

Dále je také důležité testování, ale to až ve chvíli, kdy bude produkt natolik vyvinut, aby bylo možné provádět dílčí a integrační testy.

Milník: Počáteční funkční způsobilost (Initial Operational Capability)

Milník fáze konstrukce spočívá v připravenosti a stabilitě produktu, tzv. beta-verze, pro testování na počítačích uživatele. Dále jsou tvořeny uživatelské příručky a dokumentace k produktu a je přijatelný poměr mezi plánovanými a skutečnými výdaji, které bylo nutné vynaložit na vývoj.

4. Fáze předání

Cíle

Tato fáze začíná v okamžiku, kdy je dokončeno testování a konečné nasazení produktu na pracoviště uživatele. To zahrnuje opravu všech chyb nalezených v beta-verzi a přípravu počítačů uživatele pro přijetí nového produktu. Dokončuje se práce tvorby manuálu a další dokumentace, funkčnost produktu je konzultován s uživatelem a je prováděna koncová revize. Tato fáze však může nastat i v dřívějších iteracích, například pokud není předáván produkt zákazníkovi, ale je předáván jen v rámci vývojového týmu.

Zaměření fáze předání

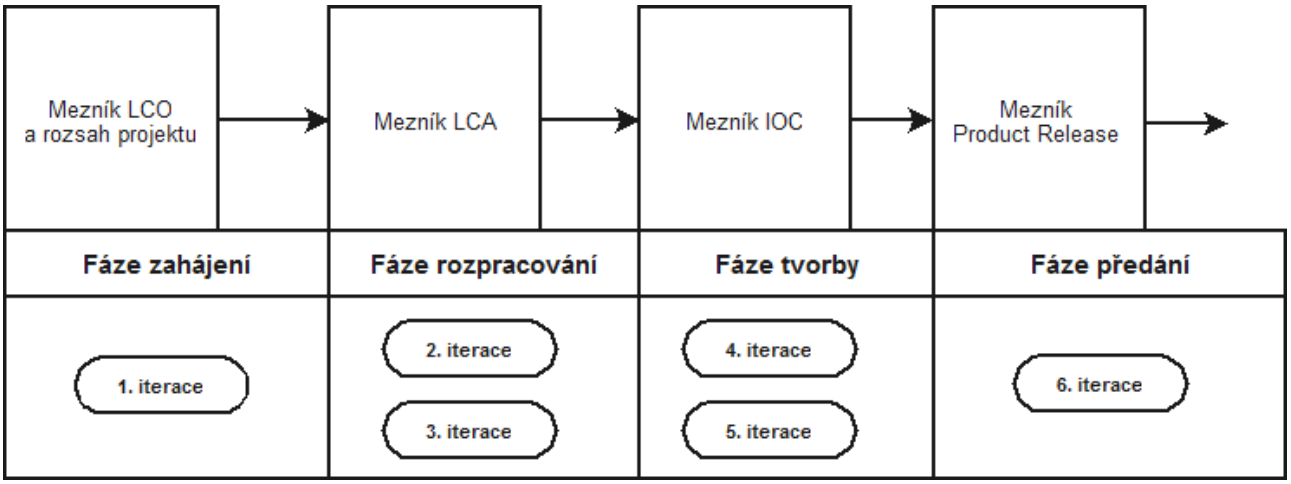
Fáze je primárně zaměřená na finální implementaci a přizpůsobení softwarového produktu na pracovišti uživatele a oprava chyb, které nebyly nalezeny při provádění beta-testů. Veškeré závady nalezené při testování beta-verzí jsou připraveny k odstranění díky vhodnějšímu návrhu. Pokud jsou v této fázi tvořeny nové požadavky a je nutná nová

analýza, není možné pokračovat a je nutné vrátit se k některé předešlé fázi.

Milník: Nasazení produktu (Product Release)

Podmínky dosažení milníku fáze předání jsou následující:

- beta-testy jsou s úspěchem dokončeny a byly provedeny nezbytné úpravy a změny
- produkt se začíná aktivně využívat
- je dohodnuta podpora produktu s uživatelem a následně je implementována



Obrázek 2: Posloupnost fází a odpovídající mezníky v metodice UP

3.5 Role

Analytik je osoba (nebo i osoby) ve vývojovém týmu, která zastupuje zákazníka a koncového uživatele. Jeho úkolem je shromáždit vstupní informace a požadavky, jaké zákazník klade na vyvíjený softwarový produkt, a dále pro ně nastavit priority, aby bylo možné začít s plánováním celého projektu.

Architekt (projektant) je zodpovědný za definici architektury softwaru, což zahrnuje i rozhodování o klíčových technických parametrech, které mohou omezit celkový design a implementaci projektu. Architekt úzce spolupracuje s projektovým manažerem při plánování projektu. Při ne příliš rozsáhlých projektech může být jedna osoba architektem a zároveň i projektovým manažerem.

Manažer projektu vede plánování projektu, spolupracuje se zákazníkem a jeho důležitým úkolem v týmu je zajistit, aby tým byl soustředěný na dokončení projektu. Zároveň je zodpovědný za analýzu rizik projektu.

Vývojář má zodpovědnost za jednotlivé části vyvíjeného softwarového produktu. Dále také provádí integraci návrhu do celkové architektury, navrhuje prototypy, které poté implementuje. V neposlední řadě se také stará o integraci komponent do větších funkčních bloků produktu.

Tester se stará o veškeré aktivity související s pracovním postupem testování. To zahrnuje nalezení, definici, implementaci a řízení nezbytných testů, stejně tak jako zaznamenávání výsledků testů a jejich analýzu.

Zákazník (investor) reprezentuje všechny osoby, které jsou nějakým způsobem zainteresovány do vývoje, tzn. mohou jistým způsobem ovlivnit výslednou podobu produktu, a nejsou členy vývojového týmu.

3.6 Závěr

Metodika UP je otevřeným standardem, který můžeme kdykoliv bezplatně použít k vývoji svých aplikací. Člení na čtyři fáze (zahájení, rozpracování, tvorba a předání) a definuje pět základních pracovních procesů (stanovení požadavků, analýza, návrh, implementace a testování).

Z metodiky UP vychází její komerční kolegyně RUP, která je obohacena o doplňkové nástroje a bonusy. Jako základní východisko pro objektový, iterativní a inkrementální vývoj softwaru je však metodika UP dostačující.

4 Scrum Development Process

4.1 Úvod

Scrum Development Process (dále jen Scrum) je jedna ze zástupců agilních metodik, které si kladou za cíl zvýšit efektivitu při vývoji softwarových produktů. Scrum, podobně jako i další agilní metodiky, využívá především výhod iterativního a inkrementálního přístupu při vývoji, zároveň se však přiklání k objektově orientovanému vidění jednotlivých části produktu, avšak při organizaci a řízení procesu se postupuje zcela novým způsobem.

Zakladateli této metodiky jsou Ken Schwaber a Mike Beedl, kteří v roce 1995 představili první náznak, jak by tato metodika měla fungovat. Hlavní cílem obou autorů bylo zlepšení produktivity procesu vývoje softwaru a přizpůsobení se měnícím se požadavkům.

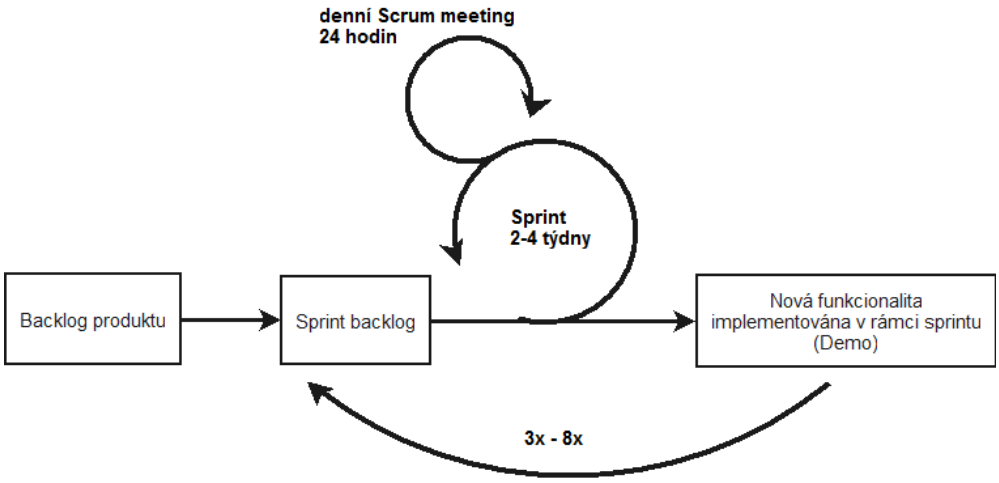
Název Scrum je převzat z ragby, kde se pod tímto pojmem označuje nahromadění se mnoha hráčů na jednom místě na hřišti, za účelem společného dotlačení míče na požadovanou pozici. Stejně jako v ragby je i při vývoji softwarového produktu cílem jeho dokončení podle představ zákazníka (požadované dotlačení míče na určené místo na hřišti).

4.2 Charakteristika

Základním členěním vývojového procesu podle metodiky Scrum jsou časové intervaly trvající přibližně 30 dnů. Tyto intervaly se nazývají Sprinty a v jednom vývoji se obvykle objeví tři až osm těchto sprintů za sebou. Metodika Scrum vychází z objektově orientovaného přístupu, tím pádem každý člen vývojového týmu odpovídá za množinu objektů s předem definovanými vlastnostmi a rozhraním.

V rámci jednotlivých sprintů nejsou definované konkrétní procesy, které mají být splněny, ale jsou pořádány každodenní schůzky, tzv. Scrum Meeting, na nichž každý člen vývojového týmu dostane přidělenou konkrétní činnost, kterou je nutné, aby splnil do dalšího Scrum Meetingu. Dále se na těchto schůzkách shrnuje, co bylo od minula dokončeno a jaké nové úkoly se objevily. Součásti jsou i procesy analýzy a designu, jelikož podle metodiky Scrum není možné v předstihu naplánovat veškerý obsah jednotlivých sprintů a proto je nutné tyto procesy řešit v rámci každodenních schůzek. Scrum Meeting v podstatě suplují centrální plánování vývoje softwarového produktu, protože tím, že probíhají každodenně a neustále je zřejmé, co je splněno a co naopak není, je vývoj mnohem flexibilnější a lze se lépe přizpůsobit změnám, které mohou ovlivnit vývoj.

Každý Sprint je zakončen předvedením výsledku zákazníkovi, tzv. Demo, na kterém je možné provést integraci k ostatním částem projektu a jeho otestování. Tím má zákazník neustálý přehled o tom, jak probíhá vývoj, které části zbývá ještě naimplementovat a které už jsou funkční.



Obrázek 3: Scrum Development Process

4.3 Podstata

Podstatou metodiky Scrum je několik pojmů, na kterých je založena:

Malý tým — Ideální tým, který používá k vývoji metodiku Scrum, má mezi třemi až šesti členy. Na větších projektech však může pracovat více Scrum týmů. Tím, že je tým tvořen méně členy vyplývá, pro jaké projekty je metodika spíše vhodná. Jedná se o ne příliš rozsáhlé projekty, na které je však kladen důraz na kvalitu a flexibilitu.

Časté revize — V rámci Scrum meetingu je prováděno zkoumání pokroků na produktu a případné změny, které bude nutné provést. Toto vše však v podstatě probíhá nepřetržitě a je to náplní práce téměř každého člena týmu.

Spolupráce — Předpokládá se, že každý člen týmu bude intenzivně komunikovat se svými kolegy, ale také to, že celý tým bude komunikovat se zákazníkem a zadavatelem práce. Požadavkem metodiky Scrum není nutná přítomnost zákazníka na pracovišti, jako je tomu u Extrémního programování, úzký kontakt a spolupráce je však zcela nezbytná. Dalším zásadním rozdílem oproti Extrémnímu programování je, že za každou množinu objektů je zodpovědná konkrétní osoba, ne jako v Extrémním programování, kde je základem společné vlastnictví. Tento rys je v metodice Scrum považován za příliš extrémní a mohlo by se zdát, že jde i o nevýhodu, avšak výhoda je v tom, že každý se snaží „své“ objekty mít co nejkvalitnější a tím je motivován k efektivnější práci.

Flexibilní harmonogram — Prvotní podmínkou používání flexibilního harmonogramu je, aby s ním byli srozuměni zákazníci a bylo jim vysvětleno, že datum dodání, at' už celého produktu či jen dema, může proběhnout dříve nebo později, než bylo původně plánováno. Toto se však může proměnit i v nevýhodu metodiky Scrum, jelikož někteří zákazníci mohou považovat za neprofesionální neustále změny v termínech dodání.

Flexibilní předměty dodání — Na konci každého sprintu je hotová určitá část z produktu, která však v různých případech může mít různé podoby.

4.4 Postup

Vývoj podle metodiky Scrum lze rozdělit do čtyř kroků:

1. **Plánování** — v rámci plánování se definuje rozsah, jaký bude mít aktuální verze, harmonogram, v jakém bude probíhat vývoj a zdroje nezbytné pro zdárné dokončení aktuální verze. Důležitým artefaktem každého plánování je tzv. backlog, kterým prakticky začíná vývojový proces. Backlog definuje úkoly a požadavky, které je nezbytné realizovat pro správnou funkcionalitu produktu. Může mít různé podoby, např. může být značen formou uživatelských příběhů, formou tabulky apod. Důležité je, že backlog může měnit pouze manažer produktu, který nejenže pozorně naslouchá celému vývojovému týmu a zákazníkovi, na jejichž základě mění jednotlivé položky, ale také třídí požadavky dle priority. V rámci plánování se jednotlivým položkám backlogu přiřadí objekty, pod které spadají, zvolí se nástroje, které budou použity pro vývoj a provede se analýza rizik. Tato analýza se však neprovádí jen při plánování, ale i na konci každého sprintu a v průběhu každého Scrum meetingu. Díky tomu lze lépe přizpůsobit náplň iterace, výslednou funkcionalitu každé verze, ale i další faktory ovlivňující samotný vývoj.

2. **Architektura a design** — v tomto kroku se vytváří nové architektury nebo modifikují ty stávající na základě nových požadavků, výsledků analýzy rizik a provádí se další, např. doménová analýza apod.

3. **Vývoj** — do tohoto kroku patří sprint, což je iterativní cyklus, při kterém se pracuje na vyvíjeném produktu. Délka jednotlivých sprintů je ovlivněna rozsahem a komplexností produktu, množstvím rizik, které byly zjištěny při analýze a v neposlední řadě počtem členů vývojového týmu. Obvykle je ale délka sprintů okolo 30 dnů.

   Na začátku každého sprintu vybere manažer projektu ty požadavky z backlogu, které mají být implementovány nejdříve. To se děje na základě různých kritérií, např. na základě toho, na co je daný sprint zaměřen, nebo zda dané požadavky spadají do stejné logické skupiny apod.

   Na vývojovém týmu potom je, aby si stanovil přibližný postup a plán této části vývoje. Zde nastává samotná náplň sprintu. Provádí se totiž návrh a implementace jednotlivých požadavků, které byly definovány v backlogu.

   Nedílnou součástí každého sprintu jsou každodenní schůzky, tzv Scrum meetings, které se účastní všichni členové vývojového týmu. Schůzka je organizována manažerem, tzv. Scrum Master, který se ptá týmu na řadu otázek, např. co se udělalo od poslední schůzky, jaké překážky se vyskytly při implementaci, na čem se bude pracovat do další schůzky apod. Každá schůzka trvá 15 až 30 minut.

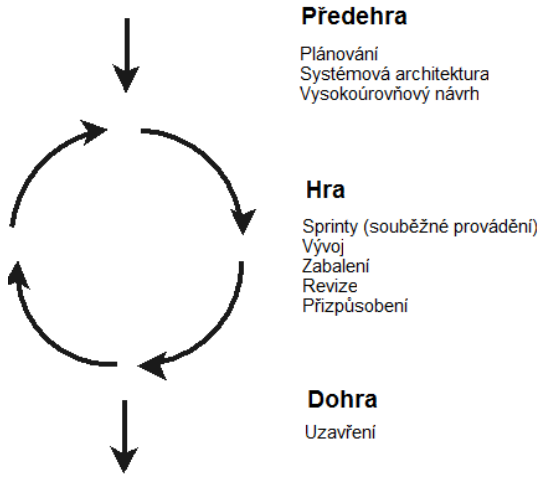
   Sprint je zakončen schůzkou, které se účastní všechny zainteresované strany, tzn. jak vývojový tým, tak i zákazník a vedení. V průběhu schůzky je předvedena nová

funkční verze. Dále se zjišťují nové požadavky, které se zaznamenají do backlogu společně s těmi, které nebyly splněny a provede se hrubý plán toho, na čem se bude pracovat v následujícím sprintu.

4. **Uzavření** — jakmile zákazník či vedení usoudí, že aktuální verze již splňuje všechny požadavky, které byly kladeny na vyvíjený produkt, nebude vývoj pokračovat dalším sprintem, ale začíná fáze uzavření. Do té doby je vyvíjený produkt stále možné a dokonce žádoucí měnit. Náplní uzavření je provést všechny integrace a testy produktu jako celku, vytváří se dokumentace a produkt se nachystá k finálnímu uvolnění a nasazení u zákazníka.

Tyto čtyři kroky, které vedou k dokončení vývoje softwarového produktu, je možné zařadit do tří fází:

- **fáze předehry** — zde patří první dva – plánování a architektura s designem
- **fáze hry** — zde jednoznačně patří vývoj
- **fáze dohry** — do poslední fáze patří také poslední krok a tím je uzavření



Obrázek 4: Fáze metodiky Scrum Development Process

Z obrázku je patrné, že nejdůležitější fází metodiky Scrum je fáze hry, tedy vývoj.

4.5 Rozdělení rolí

Jednotlivé role všech zainteresovaných při vývoji softwarového produktu lze rozdělit na dvě skupiny. První z nich jsou **Pigs**, což jsou pouze členové týmu a druhou **Chickens**, zde patří všichni, co nejsou členy vývojového týmu, především zákazníci.

Podstatou tohoto rozdělení je to, že při Scrum meetingu mají hlavní slovo pouze Pigs, zatímco Chikens se mohou účastnit, ale smí jen pozorovat a nijak do této schůzky nezasahovat.

Obecně lze Pigs, tedy tým rozdělit ještě na vedení a vývoj.

Vedení je skupina vedena produktovým manažerem a jejich hlavní úkolem je definice obsahů a termínů jednotlivých verzí. Z toho plyne, že pracují především z backlogem a analýzou rizik. Dále řídí a směrují průběh vývoje a kontrolují, zda tým vykazuje pokroky a odhalují případné odchylky od předběžných plánů.

Druhou skupinou Pigs je **Vývoj**. Obvykle se jedná o skupinu 3 až 6 osob, kde jsou zastoupeni jak programátoři a dokumentátoři, tak i osoby zodpovědné za řízení kvality. Pokud je při vývoji produktu metodikou Scrum nutná početnější skupina, je doporučována vytvořit více menších týmu, které mohou být zodpovědné každý za jednu skupinu funkcí, nebo každý za jednu vrstvu systému. Záleží zda je zvoleno dělení na základě funkcionality, nebo na základě systémového dělení.

4.6 Závěr

Scrum je zástupce agilní metodiky, která zlepšuje produktivitu a pružnost vývoje softwarového produktu díky novému přístupu k organizaci práce v týmu.

Vývoj pomocí Scrum začíná úvodní plánovací fází, v jejímž rámci se vytvoří a klasifikuje seznam požadovaných úkolů (Backlog), stanoví hrubý plán projektu a první návrh architektury. Následují vývojové sprinty, což jsou několikatýdenní vývojové části. Cílem každé z nich je uvolnění nové verze produktu (tzv. Demo) a představení nové funkčnosti.

5 Extrémní programování

5.1 Úvod

Extrémní programování (dále jen XP) je lehká a účinná agilní metodika pro vývoj softwarových produktů v malém až středně velkém týmu, při kterém je kladen důraz na nutnost se přizpůsobit měnícímu se zadání, nebo zadání, které není na začátku vývoje až tak jasné. XP se snaží o snížení projektových rizik, zlepšení reakce na změny a zvýšení produktivity práce během celého životního cyklu vývoje softwaru.

Vznik metodiky XP se uvádí v roce 1999, kdy známý expert na vývoj softwaru Kent Beck představil její základní myšlenky. O tři roky dříve se však objevují první zmínky o jejím vývoji. Autor a jeho tým totiž nebyli schopni ani po roce usilovné práce vyvinout systém týkající se mezd v Chrysler Corp. nastavenou metodikou. Ta nebyla schopna přizpůsobovat se průběžným změnám. Proto začal pracovat na návrhu metodiky, která by měla umožnit flexibilnější vývojový proces.

Od ostatních metodik se odlišuje především těmito vlastnostmi: nepřetržitá a konkrétní zpětná vazba, rychle vytvořený celkový plán, schopnost flexibilně určovat termíny a využívat zautomatizovaných testů.

Dále se mírně liší v proměnných, které jsou brány na zřetel při vývoji softwarového produktu (tzv. řídící proměnné) (viz obrázek 1) a navíc přidává čtvrtou proměnnou.

- KVALITA (odpovídá funkcionalitě na obrázku 1): Pokud se rozhodneme snížit kvalitu softwaru, který vyvíjíme, může to sice vést ke krátkodobému zisku, ale náklady, které musíme vynaložit, at'už se jedná o počet programátorů ve vývojovém týmu, či firemní nebo technické, se poté enormně zvýší.

- ČAS: Prodloužíme-li čas na vývoj softwaru, můžeme zvýšit kvalitu a rozšířit i zadání. Velké prodloužení však může také vést ke snížení kvality, jelikož zpětná vazba od produktu v provozu má pro nás daleko větší cenu, než jakékoliv jiné druhy zpětné vazby. Je-li příliš málo času na vývoj produktu, je to znát i na kvalitě a šíři zadání.

- NÁKLADY (odpovídá zdrojům na obrázku 1): Více financí při vývoji nám může na jednu stranu sice zlepšit vyhlídky, ale na stranu druhou může příliš mnoho financí způsobit více problémů než užitku. Vloží-li se do projektu příliš malé množství financí, je zadání zákazníka téměř neřešitelné.

- ŠÍŘE ZADÁNÍ: Jedná se o novou proměnnou, kterou XP přidává ke třem tradičním a říká, které funkce jsou pro zákazníka klíčové a které je naopak možno, při výskytu problémů, odložit v implementaci. Pokud zvolíme menší šíři zadání, přitom jsme však stále schopni vyřešit problém zákazníka, docílíme vyšší kvality za méně času i finančních prostředků.

V tomto modelu se vývoj softwarového produktu odehrává tímto způsobem: vnější síly, což jsou zákazníci a manažeři, si stanoví vstupní hodnoty tří z těchto proměnných, které chtějí řídit. Zbývající čtvrtou má poté na starost vývojový tým. Pokud je jedna ze stran

nespokojená s výslednou hodnotu této čtvrté proměnné, je nutné upravit vstupní hodnoty zbylých tří proměnných. Pokud se zákazníci či manažeři budou snažit nastavit hodnoty všech čtyř proměnných, tedy kvalitu, čas, náklady i šíři zadání, povede to jedině k tomu, že vývojový tým bude pod velkým tlakem a v důsledku toho dojde ke snížení kvality celého vyvíjeného produktu a v mnoha případech také ke zdržení.

Základním problémem, který se snaží XP vyřešit, je riziko. A to především riziko zpoždění harmonogramu, zrušení projektu, riziko krachujícího systému, míry poruchovosti, nepochopení nebo změny zadání a riziko přemíry funkcí.

5.2 Podstata

Řešení těchto rizik je v konečném důsledku podstata Extrémního programování.

Zpoždění harmonogramu
XP je charakteristické krátkými cykly pro uvolnění nové verze, které trvají nejvýše několik měsíců. V rámci nové verze se dodržují jednotýdenní až čtyřtýdenní iterace, čímž je zajištěna velmi dobrá zpětná vazba od zákazníka. Aby se případné problémy projektu mohly vyřešit během iterace, jsou stanoveny jednodenní až třídenní úkoly. V XP má také každá požadovaná funkce vyvíjeného softwarového produktu svou prioritu. Z toho je zřejmé, které funkce je nutné naimplementovat přednostně a které se objeví až v další verzi produktu, to znamená mají nižší prioritu. [1]

Zrušený projekt
Vývoj softwarového produktu pomocí XP požaduje od zákazníka definici nejmenší množinu funkcí, která má obsahovat možné verze produktu a jež má pro něj největší smysl a význam. Tím se minimalizuje to, co by se mohlo pokazit ještě před uvedením do provozu a zároveň se hodnota daného softwarového produktu maximalizuje. [1]

Krachující systém
Jedním ze základů metodologie XP je vytváření a udržování kompletní sady testů, které se spouštějí jednou před a podruhé po každé změně, která se provede na vyvíjeném produktu, což se může dít i několikrát za den. [1]

Míra poruchovosti
XP testuje celky produktu jak z pohledu programátorů, kteří píší testy pro jednotlivé funkce, tak i z pohledu zákazníků, kteří se velkou mírou podílejí na tvorbě testů pro celé funkční části softwaru. [1]

Nepochopení zadání
XP požaduje od zákazníka, aby se integroval do vývojového týmu a tím se specifikace a požadavky na projekt neustále během vývoje upřesňují. [1]

Změny zadání
Při zkracování cyklů, kdy se uvolňují nové verze produktu, je menší pravděpodobnost,

že zákazník bude požadovat zásadní změny, než kdyby doba cyklu byla příliš dlouhá. Pokud zákazník požaduje změnu ještě nedokončené funkce produktu, stačí předefinovat priority jednotlivých funkcí. [1]

Přemíra funkcí
Je kladen velký důraz na to, aby se řešily nejprve úkoly s nejvyšší prioritou. [1]

Z výše uvedených částí plynou základní hodnoty a postupy Extrémního programování.

5.3 Základní hodnoty

Základními hodnotami, kterými se XP řídí a na kterých staví, jsou komunikace, jednoduchost, zpětná vazba, odvaha a pod těmito čtyřmi leží ještě pátá podprahová – respekt.

1. **Komunikace**
   Při výskytu problémů či zdržení ve vývoji softwarového projektu je velice pravděpodobné, že prvotní příčinou je špatná komunikace. Většinou se jedná o to, že jeden z členů týmu zapomene nebo neshledá za důležité, sdělit ostatním informaci týkající se změny či úpravy některých částí projektu.

   V rámci XP se používá celá skupina postupů, které se bez komunikace zkrátka neobejdou, jedná se například o párové programování, odhadování termínů a další. Mimo to je v XP zastoupena přímo role, při které má daný aktér také za úkol pozorovat a obnovovat dobrou komunikaci mezi všemi články vývojového týmu. Jde o tzv. Kouče.

2. **Jednoduchost**
   Klíčovým je schopnost nemyslet na předpokládanou budoucnost, ale raději udělat jednoduchou věc dnes, což je v mnoha případech i ekonomičtější, a v budoucnu zaplatit o něco více za případnou změnu, která však také nemusí ani nastat, než udělat dnes složitější věc, která se ovšem nemusí za rok vůbec využít.

   Jednoduchost a komunikace se doplňují. Čím lepší je komunikace v týmu, tím je jasnější, co je potřeba udělat a naopak víme, co se dělat vůbec nemusí. Čím jednodušší je vyvíjený produkt, tím menší a úplnější je vzájemná komunikace.

3. **Zpětná vazba**
   Zpětná vazba v XP je realizována několika metodami, přičemž asi nejdůležitější je testování. To probíhá v různých fází vývoje. Programátoři, jakmile dostanou nová zadání od zákazníků, ihned odhadují termíny dokončení a tím mají zákazníci okamžitou zpětnou vazbu. Dále programátoři píší jednotkové testy pro veškerou logiku náchylnou na chyby. Během pár minut mají zpětnou vazbu o správnosti či nesprávnosti jejich kódu. Člověk, zastupující roli Stopaře, čekající na termín dokončení úkolu předává týmu zpětnou vazbu o tom, zda pracují dle plánu, který si odhadli na začátku. Zpětná vazba funguje také v delším časovém úseku. Zákazníci, za pomoci Testerů, píši testy funkcionality implementovaných úkolů. Mají tak konkrétní zpětnou vazbu o stavu celého projektu. Další strategií je uvedení do provozu

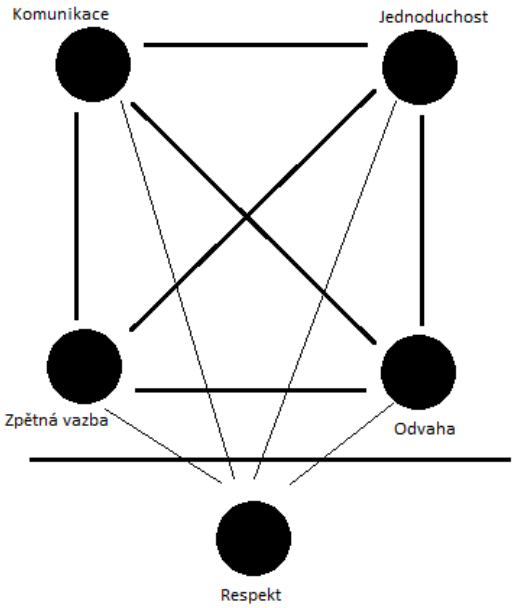
těch nejhodnotnější zadání co nejdříve, tím se programátorům vrátí zpětná vazba o kvalitě svých schopností.

V XP platí: čím více zpětné vazby, tím lépe, jelikož je pravděpodobnější úspěšné dokončení celého projektu. Také je snadnější komunikace, protože v případě, že má někdo z týmu námitku proti zdrojovému kódu a dodá k tomu test, který potvrdí jeho opodstatnění, máme opět konkrétní zpětnou vazbu, na kterou můžeme reagovat. S prací bychom měli být hotovi teprve tehdy, až úspěšně proběhnou všechny testy. Spoustu zpětnovazebních informací získáme pouze dobrou komunikací s ostatními. Z toho plyne, že komunikace se zpětnou vazbou doplňuje, podporuje a v mnoha případech tvoří celek.

4. **Odvaha**
   Pokud tým dojde k přesvědčení, že z jakékoliv příčiny již nevede cestu kupředu, je nezbytné dané příčiny odstranit a to za jakoukoliv cenu, i v případě, že by to znamenalo zahodit větší část již napsaného zdrojového kódu, či přepracovat celý návrh i s architekturou projektu. To jsou odvážné kroky, protože jsme stále vázání na dodržení všech podmínek zadání, ale bez dostatečné dávky odvahy je použití metodiky XP téměř vyloučeno.

   Komunikace podporuje odvahu, protože nám umožňuje experimentovat sice s vyšším rizikem, ale také ziskem. Z toho také plyne, že odvaha je podporována jednoduchostí, protože jen s jednoduchým systémem si můžeme dovolit být více odvážnější. A také to platí naopak: Odvaha podporuje jednoduchost, protože jakmile se vyskytne možnost na zjednodušení systému, zkusíme to. A nakonec i zpětná vazba podporuje naší odvahu, neboť ji máme daleko větší při rozsáhlejší změně části zdrojového kódu, protože správnost si můžeme okamžitě ověřit pomocí testů.

5. **Respekt**
   Pod povrchem těchto čtyř hodnot existuje pátá – respekt. V celém vývojovém týmu je nutné, aby každé člen projevoval zájem o to, co dělají nebo právě vyvíjejí jeho kolegové. Tím také zjistí, jak by mohl být on sám nápomocen ostatním a jak by mohl zefektivnit celý proces vývoje. Pokud by to tak nebylo a každý člen týmu by hleděl jen na svou práci, bylo by Extrémní programování nepoužitelné. Totéž však neplatí jen v XP, ale i v ostatních metodikách.



Obrázek 5: Vzájemný vztah pěti hodnot Extrémního programování

5.4 Pracovní postupy

Je definováno dvanáct konkrétních postupů, které by měli vést k vytváření kvalitních softwarových produktů pomocí metodiky Extrémní programování.

Plánovací hra

Postup, který je určený k plánování projektu a k plánování toho, co budou jednotlivé iterace a fáze vývoje obsahovat, se nazývá Plánovací hra. Měli by se jí účastnit všechny strany, tzn. zákazníci společně s vedením a vývojový tým.

Pravidla hry jsou následující:

Cílem hry je maximalizovat hodnotu produktů, který tým vyvíjí. Z této hodnoty musíme však odečíst náklady, které jsme vynaložili na jeho vývoj a riziko, které podstupujeme při jeho tvorbě.

Strategie týmu je taková, že se snaží uvést do provozu pro zákazníka nejcennější funkce, ale pouze s využitím programovacích a návrhových postupů, jež snižují riziko.

Hracími kameny a podstatou této hry je tvoření Karet zadání. Ty představují základní informace o požadavcích kladených na vyvíjený softwarový produkt.

Hráči Plánovací hry je Vývoj, tvořený všemi zodpovědnými za implementaci, a Vedení složené ze všech, kteří rozhodují o jednotlivých funkcích produktu, tedy i koncoví zákazníci.

Dalším pravidlem jsou Tahy. Tahy můžeme použít ve třech fázích:

1. **Průzkum** — cílem této fáze je oběma hráčům přiblížit to, co je od produktu očekáváno na konci vývoje. Patří zde tyto tahy:

Napsání zadání: Vedení sepíše své představy na listy, obsahující název zadání a účel, co by měl vyvíjený produkt dělat.

Odhad zadání: Vývoj odhadne ideální čas, který potřebuje k implementaci daného zadání. Pokud Vývoj není schopný odhadnout potřebný čas, požádá Vedení o upřesnění zadání.

Rozdělení zadání: Pokud se stane, že Vedení zatím neví celé zadání, nebo chce určitou část nechat naimplementovat přednostně, může se zadání rozdělit na několik samostatných a ty řešit postupně.

2. **Závazek** — v této fázi by Vedení mělo stanovit šíři zadání a datum uvolnění další verze produktu. Vývoj se poté zaváže k dodržení těchto podmínek. Závazek lze rozdělit do čtyř tahů:

   Třídění dle hodnoty: Vedení roztřídí zadání na tři hromádky – zadání, bez kterých by systém nefungoval, zadání, která mají nižší prioritu, přesto však představují podstatu produktu a zadání, které by výsledný produkt nějak obohatila.

   Třídění dle rizika: Vývoj vytřídí zadání, která nemůže přesně odhadnout, dále zadání, která jsou odhadnutelná celkem dobře a nakonec zadání, která nelze odhadnout vůbec. Tím vzniknou opět tři hromádky zadání.

   Nastavení rychlosti: Vývoj se dohodne, jak rychle je schopný programovat v ideálním čase za kalendářní měsíc a to sdělí Vedení.

   Volba šíře zadání: Vedení vybere sadu karet zadání pro danou verzi a to na základě určení data, kdy má být vývoj hotov, nastavení rychlosti, nebo výběrem karet a výpočtem data.

3. **Řízení** — účelem řídící fáze je aktualizace plánu danou poznatky Vývoje a Vedení. Opět se skládá ze čtyř tahů:

   Iterace: Na začátku každé iterace, která obvykle trvá jeden až tři týdny, vybere Vedení nejcennější zadání, která se mají implementovat v dané iteraci. Výsledkem po první takové iteraci by měl být běhu schopný základ vyvíjeného produktu.

   Přehodnocení: Pokud Vývoj zjistí, že správně neodhadl rychlost implementace, může od Vedení zjistit, jaká je nejcennější sada zadání, která je nutná naimplementovat v aktuální verzi za dané nové rychlosti implementace.

   Nové zadání: Jestliže Vedení přijde na nové zadání, které je nutné implementovat v aktuální verzi, sepíše jej a Vývoj ho odhadne. Na Vedení potom je, aby rozhodlo o odstranění jiného zadání s ekvivalentním odhadem z plánu a vloží místo něj toto nové.

   Nový odhad: Může se stát, že plán již neposkytuje přesný harmonogram vývoje a je na Vývoji, aby znovu odhadl zbývající zadání a nastavení rychlosti.

Princip je následující: zákazník sepíše všechny funkční požadavky na vyvíjený produkt, každý z nich se rozepíše do tzv. uživatelského příběhu, ve kterém je daná funkce popsána jazykem zákazníka. Následně si tyto požadavky projde vývojový tým a určí, za jak dlouho

a za jakou cenu je schopný tyto funkce vytvořit. Poté se opět zákazník rozhoduje, kterým požadavkům dá vyšší prioritu a tím i přednostní vytvoření týmem. Na základě toho se vytvoří plány jednotlivých verzí a karty zadání. Tým společně se zákazníkem tak určují, jak docílit maximální efektivity, minimálních finančních prostředků a uvedení prvotních verzí produktu do provozu za co nejkratší čas.

Malé verze
Každá nová verze by měla být co nejmenší, přitom však kompaktní a měla by odrážet jen ta nejcennější zadání zákazníka. Nemělo by doházet k tomu, že vývojový tým chce uvolnit novou verzi za každou cenu a co nejdříve. Mohlo by se pak stát, že je například implementována jen poloviční funkcionalita dané verze. Proto je lepší plánovat po menších časových úsecích.

Metafora
Celý vývoj pomocí metodiky XP je veden jedinou metaforou. Je to jakýsi příběh o tom, jak má celý produkt fungovat. Slova používána k identifikace technických atributů by měla být vybírána pouze ze zvolené metafory a měla by jí rozumět každá strana, která se podílí na vývoji. Metafora v XP nahrazuje většinu z toho, co si v ostatních metodikách představíme pod pojmem „architektura“. Příkladem metafory může být abstrakce „výpočet důchodu je jako tabulka“. Všem je jasné, že výpočet ve skutečnosti není tabulkou, ale pomůže jim to sjednotit pohled na celý problém a to je cílem metafory.

Jednoduchý návrh
Správný návrh by měl v kterémkoliv okamžiku splňovat tyto pravidla: všechny testy jsou funkční, návrh neobsahuje duplicitní logiku, programátoři jsou seznámení se všemi důležitými záměry, návrh má co nejmenší počet tříd a metod. Hlavním úkolem je, aby nejjednodušší návrh fungoval v aktuální verzi a jakákoliv komplikovanost byla odstraněna ihned po odhalení. Všechny nové prvky se implementují teprve tehdy, kdy jsou skutečně nezbytně nutné. Jedná se o opak toho, co je běžné: „implementujeme pro dnešek, navrhujeme pro zítřek“. Metodika Extrémního programování však preferuje myšlenku: „vložíme tam to, co potřebujeme, když to potřebujeme“.

Testování
Aby nedocházelo k tomu, že napsaný zdrojový kód je nekvalitní, je nutné testování.

Testy jednotek psané programátory musí vždy probíhat na 100%. Tyto testy se spouští před integrováním nové části zdrojového kódu i po ní. Jestliže je problém při nějakém testu, pro nikoho z týmu není nic důležitějšího než jeho odstranění.

Zákazníci píši testy po jednotlivých příbězích. Vždy si určí, co musí otestovat, aby zkontrolovali, že je dané zadání splněno. Všechny takovéto scénáře se za asistence testerů přemění na tzv. testy funkcionality. Tyto testy nemusí nutně probíhat na 100%, ale s blížícím se koncem vývoje by se měli tomuto číslu blížit.

V Extrémním programování neexistuje žádná funkce programu, která by neprošla testem.

Refaktorizace

Refaktorizace je postup jak zpřehlednit a zjednodušit produkt beze změn v jeho chování tak, aby byl srozumitelnější pro všechny členy vývojového týmu. Při vývoji může docházet ke dvěma druhům refaktorizace:

Refaktorizace, u které je podmínkou, že se nemění stávající kód, ale pouze se doplňuje o nové funkce, na které se potom provedou testy a používá se při přidávání nové funkcionality.

Principem druhé refaktorizace je, že se nepřidávají žádné nové schopností, jen se upravuje strukturu současného kódu.

Je nutné si neustále uvědomovat, která refaktorizace se zrovna používá. Refaktorizovat lze před změnou programu na základě toho, zda existuje způsob, jak upravit program, aby přidání nové funkce bylo jednodušší, nebo po změně programu, pokud zjistíme, že existuje možnost, jak program zjednodušit.

Neměli bychom refaktorizovat slepě, ale jen v případě, kdy je to vyžadováno. Správně prováděnou refaktorizací lze dosáhnout toho, že jsou odstraněny duplicity, zdokonaluje se komunikace, vylepšuje se návrh, optimalizují se algoritmy a mnoho dalšího.

Párové programování

Veškerý kód, který vznikne pomocí metodiky XP je psán dvěma programátory, kteří sdílejí jeden počítač s jedním monitorem, klávesnicí i jednou myší. V každém páru jsou rozděleny dvě role.

Jeden programátor, který je zrovna u klávesnice a myši přemýšlí o nejlepším způsobu implementace daného problému v tomto místě. Zatímco druhý z partnerů přemýšlí o problému s ohledem na strategii a její dodržování, zda toto řešení bude fungovat, jestli existují ještě nějaké testy, které by po integraci nové části kódu nemusely fungovat, či jestli je ještě jiný způsob, jak zjednodušit celý systém.

Tvoření páru je dynamické. Pokud dva lidé jsou spolu v páru ráno, neznamená to, že v průběhu odpoledne se páry nemohou prostřídat.

Párové programování také omezuje výskyt některých zlozvyků. Mnozí programátoři časem vynechávají psaní testů, odkládají refaktorizaci nebo integraci. Pokud se však dívá jeho partner, je pravděpodobnost výskytu těchto nešvarů daleko menší. Tím však není řečeno, že pár jako celek chyby nedělá, naopak a právě proto je v týmu potřeba Kouče.

Párové programování v XP funguje velice dobře díky neodmyslitelné komunikaci. Protože se páry neustále mění, jakákoliv důležitá informace se uvnitř týmu rychle šíří a tím se zlepšují dovednosti a získávají se nové a nové zkušenosti.

Společné vlastnictví

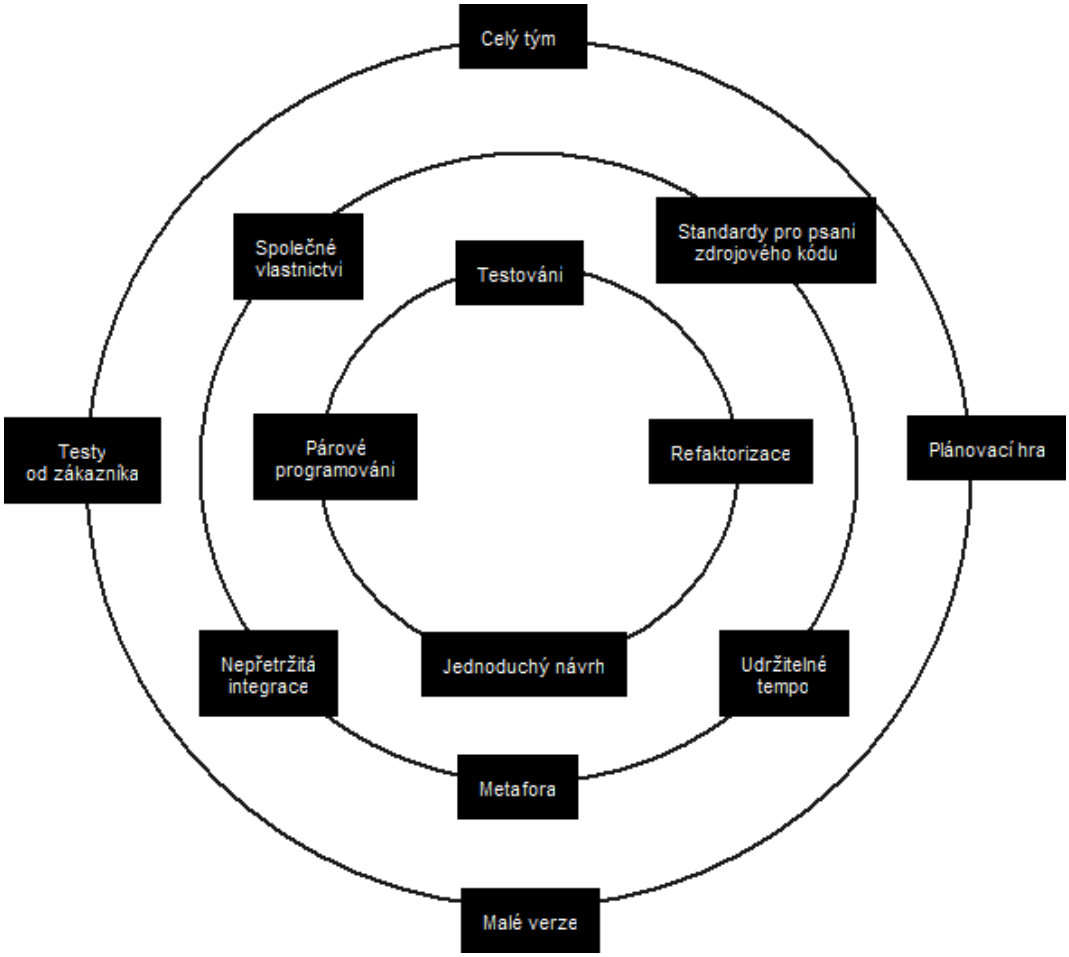
Na rozdíl od jiných metodik v Extrémním programování je povinen kdokoli, kdo vidí příležitost k přidání kódu do libovolné části zdrojového textu, tak učinit a to kdykoli. Z toho jasně vyplývá to, že všichni členové vývojového týmu mají odpovědnost za celý systém. Každý z členů sice nezná jednotlivé části zdrojového kódu stejně dobře, alespoň něco ví o každé.

Nepřetržitá integrace
Nové části zdrojového kódu jsou integrovány několikrát, nejméně však jedenkrát za den, pokaždé, když je vyřešena další funkcionalita vyvíjeného produktu. Je doporučeno mít vyhrazený jeden počítač pouze na integrování. Pár, který chce integrovat své řešení, musí také provést všechny testy, aby produkt po integraci pracoval 100% správně. Pokud to pár nezajistí, nezbývá než své řešení zahodit a začít nanovo.

Čtyřicetihodinový pracovní týden
Extrémní programování se řídí myšlenkou, že dlouhodobé přesčasy vedou k přetěžování lidských zdrojů a ta ústí v neefektivitu. Dále se dodržuje to, že je nepřípustné pracovat přesčas dva týdny za sebou. Protože přetěžování lidských zdrojů je známkou nesprávného řízení projektu.

Zákazník na pracovišti
Součástí týmu by se po dobu vývoje produktu měl stát i skutečný zákazník. Skutečný ve smyslu někdo, kdo bude s produktem skutečně pracovat, jakmile ho tým nasadí do provozu. Tento zákazník by měl být v týmu po celou dobu, tzn. 40 hodin týdně. To však může být pro jeho zaměstnavatele příliš drahé, ale na druhou stranu by mu to měla vynahradit hodnota lépe fungujícího a dříve dodaného produktu. Zákazník by měl být schopen odpovídat na otázky programátorů a měl se podílet na tvorbě testů, řešení sporů a určování priorit.

Standardy pro psaní zdrojového textu
Při vývoji pomocí Extrémního programování nevznikají prakticky žádné dokumentace a specifikace. Základním nositelem informací je zdrojový kód, proto je velice důležité, aby byl čitelný, strukturovaný a dobře srozumitelný pro všechny členy týmu. Toho je možné docílit jedině tak, že všichni programátoři budou dodržovat stanovené pravidla.



Obrázek 6: Pracovní postupy Extrémního programování

5.5 Činnosti

Dle Extrémního programování existují z praktického hlediska čtyři činnosti vývoje: Testování, Psaní zdrojového kódu, Poslouchání, Navrhování.

Testování
Testování probíhá průběžně, obvykle před a po nějaké změně ve zdrojovém kódu. Novou část zdrojového kódu nelze integrovat bez toho, aby proběhl úspěšný test na tuto část. Testy jako takové můžeme rozdělit do dvou sad: první takovou sadou jsou testy jednotek psané programátory, kteří je píši, aby byli přesvědčení o tom, že vyvíjená část produktu pracuje tak, jak si mysleli, že by měla pracovat a aby toto přesvědčení mohli sdílet i ostatní kolegové ve vývojovém týmu. Druhou sadou testů jsou testy funkcionality specifikované zákazníky, které je mají přesvědčit o tom, že vyvíjený produkt jako celek pracuje dle jejich požadavků a představ. Veškeré testy jsou izolované, tudíž neovlivňují ostatní testy

a automatické, to znamená, že jejich výsledky jsou nezávislé a neovlivnitelné, proto jsou testy tak důležitým stavebním prvkem v metodice XP.

Psaní zdrojového kódu
Zdrojový kód je cílem, ale zároveň i metodou Extrémního programování. Psaní zdrojového kódu začíná bezprostředně po napsání testu pro příslušnou část kódu a bez něj nemůže existovat vyvíjený softwarový produkt. Zdrojový text lze využívat také ke komunikaci, k vyjádření záměrů, popisu algoritmů a podobně.

Poslouchání
Neodmyslitelnou schopností vývojářů je naslouchání zákazníkům i svým kolegům, jelikož bez správného naslouchání není možné dobře implementovat řešení daného problému.

Navrhování
Abychom se při vývoji metodikou XP nedostali do „slepé uličky“, ze které není efektivní „cesty ven“, je nezbytné provádět návrh. To je jediný způsob, jak se tomu vyhnout – z možnosti se stává nutnost. Navrhování je vytvoření struktury, která organizuje logiku v systému. Dobrý návrh je vytvořen tak, že změna v jedné části produktu není nutnou podmínkou ke změnám i v jiných částech. Způsob, jakým se v XP docílí dobrého návrhu, je zcela odlišný od návrhů v ostatních softwarových procesech. Návrh je každodenní pracovní náplní všech programátorů.

5.6 Závěr

Extrémní programování je flexibilní agilní metodikou určenou především pro menší vývojové týmy.

Tato metodika klade důraz na provázanost fází návrhu a implementace, na inkrementální vývoj probíhající v krátkých iteracích a na úzký kontakt se zákazníkem. V XP neprobíhá plánování architektury příliš dopředu, jelikož se očekává, že než bude implementována, změní se požadavky.

6 Srovnání softwarových procesů

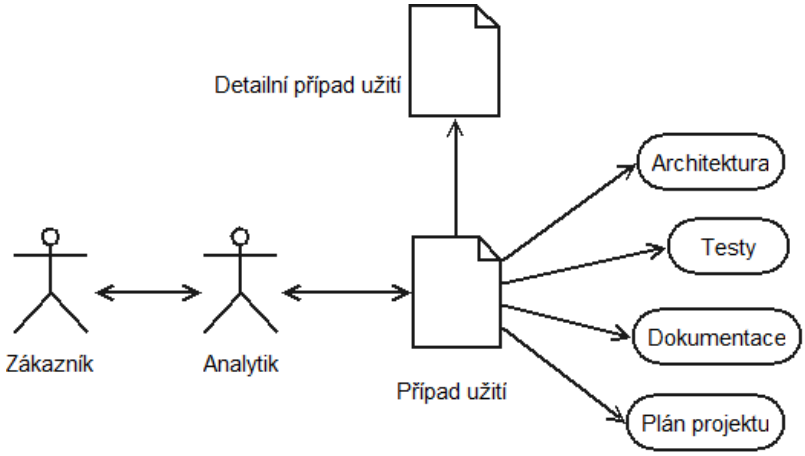
6.1 Specifikace požadavků čitelné pro zákazníka

Cílem této specifikace je zachycení všech důležitých funkcí, vlastností a rysů, které má splňovat vyvíjený softwarový produkt.

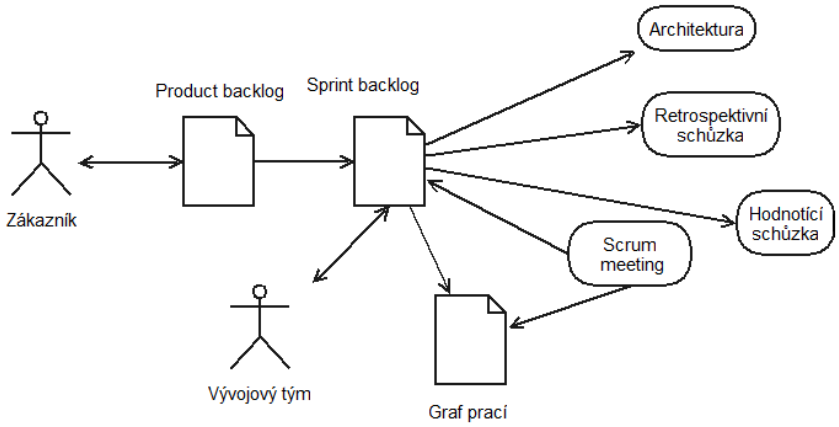
Metodika UP pro tuto specifikaci používá pojem *případ užití*. Ten se používá k obecnému porozumění chování vyvíjeného softwarového produktu, k návrhu částí, které jsou nutné k dosažení požadovaného chování, k identifikaci testů, které bude nutné provést, k zhodnocení a plánování další práce a v neposlední řadě k psaní uživatelských příruček a dokumentace. Případ užití je vstupem pro tvorbu testů, pro návrh a implementaci řešení, pro nastínění architektury vyvíjeného softwarového produktu a pro navržení plánu dané iterace. Výstupem je detailní scénář případu užití, rozpoznání a nástin požadavků kladených na softwarový produkt. Zodpovědnost za změny týkající se případu užití nese *Analytik*, jelikož úzce spolupracuje se zákazníkem a je s ním v neustálém kontaktu. Může také jako jediný provádět změny.

Metodika Scrum používá ke specifikaci požadavků pojem *Product backlog*. V něm je zákazníkem – zadavatelem projektu specifikováno, jak by měl vyvíjený produkt vypadat a jak by se měl chovat, to vše seřazeno podle priority. Na vývojovém týmu poté je, aby na *plánovací schůzce* z něj vybral ty funkce a vlastnosti, které budou implementované v nadcházejícím sprintu. Tím vytvoří tzv. *Sprint backlog*, který je podmnožinou product backlogu a zároveň jeho výstupem. Zodpovědnost za tvorbu a změny provedené v sprint backlogu má celý vývojový tým. Backlog je vstupem k provedení revize, zhodnocení celého sprintu, výstupem pro každodenní Scrum meetingy a k naplánování průběhu sprintu. Během sprintu *Scrum master* dohlíží na to, aby Sprint backlog neustále odrážel skutečnost, tzn. zjišťuje, které úkoly již byly splněny a které ještě zbývá udělat a jak dlouho vše asi bude trvat. Výsledky zaznamenává do *grafu prací (Sprint Burndown Chart)*.

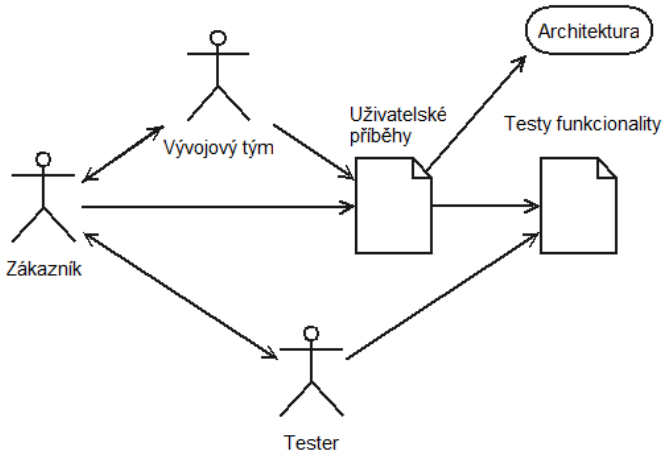
Metodika XP specifikaci požadavků nazývá *uživatelskými příběhy*. Tvorba těchto uživatelských příběhů probíhá následovně: členové vývojového týmu se ptají zákazníka na otázky, týkající se budoucího softwarového produktu, tak, aby si udělali představu o tom, co od něj zákazník čeká a požaduje. Zodpovědnost za obsah a možnost úpravy uživatelských příběhů je na zákazníkovi. Uživatelské příběhy jsou vstupem pro tvorbu zákaznických testů, které obvykle ve spolupráci přímo se zákazníkem tvoří *Tester*. Výstupem jsou vytvořené uživatelské příběhy.



Obrázek 7: Specifikace požadavků v metodice UP



Obrázek 8: Specifikace požadavků v metodice Scrum



Obrázek 9: Specifikace požadavků v metodice XP

Z obrázků 7, 8, 9 je vidět, že hlavním aktérem, při specifikaci požadavků kladených na budoucí produkt, je u všech tří metodik samotný *zákazník*. Ten společně s vývojovým týmem vytváří určité artefakty (u UP je to *Případ užití*, u Scrum *Product backlog* a u XP *Uživatelské příběhy*), od kterých se poté odraží samotný návrh a architektura vyvíjeného softwarového produktu.

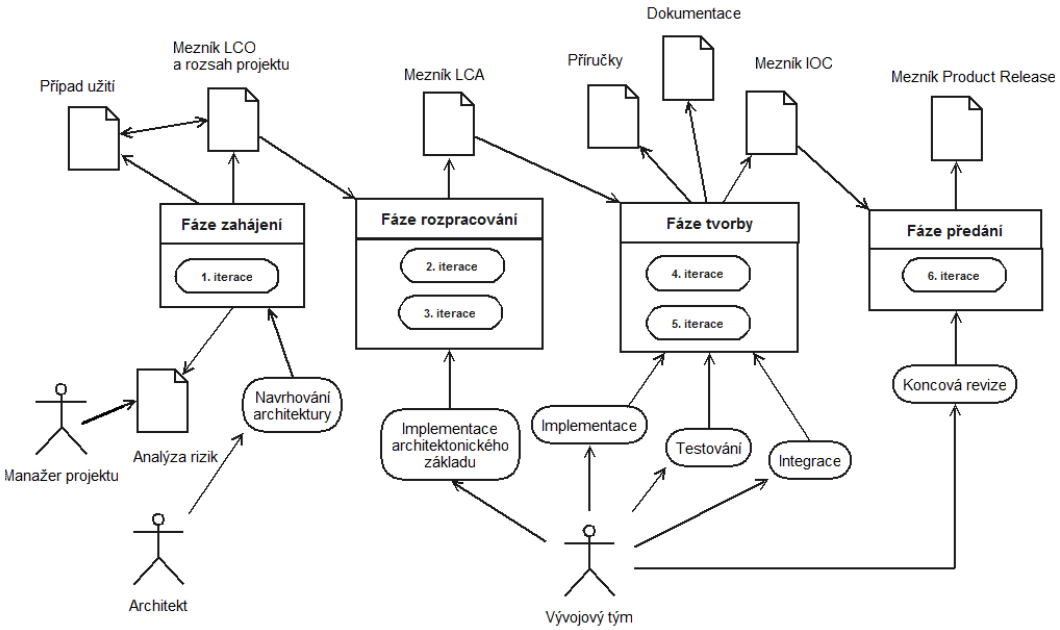
6.2 Časové dělení vývojového procesu

Celý vývojový proces se dělí do časových intervalů. Ty se dále ještě mohou členit na fáze, ve kterých probíhají různé pracovní postupy, jako je plánování, analýza, návrh, implementace, testování, integrace a předání. Po uplynutí pevně stanoveného časového intervalu je předán výsledek intervalu k zhodnocení buď zákazníkovi, nebo jen v rámci týmu. Následně jsou na něm prováděny úpravy a změny, aby byly splněny všechny požadavky, které na něj zákazník klade.

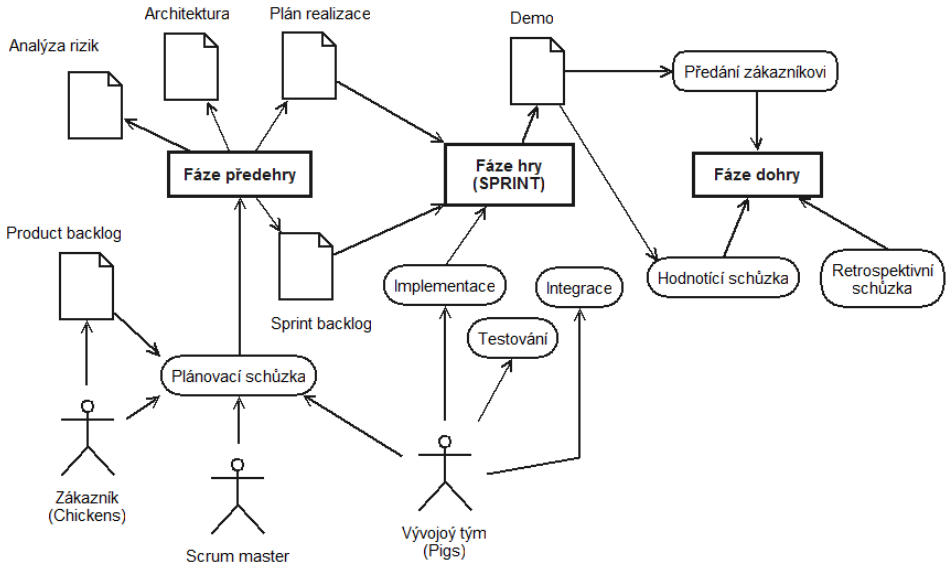
Metodika UP se dělí na *čtyři fáze*, v každé probíhá jedna nebo více iterací, které trvají maximálně 2-3 měsíce. Kolik iterací proběhne v rámci fáze záleží na rozsahu vyvíjeného softwarového produktu. Aby bylo možné přejít do další fáze, je nutné dosáhnout *mezníku (milníku)*.

V rámci vývoje metodikou Scrum třikrát až osmkrát proběhne *Sprint* (opět v závislosti na rozsahu softwarového produktu), což je časový interval trvající 2-4 týdny, v jehož průběhu projde vyvíjený produkt všemi pracovními postupy metodiky.

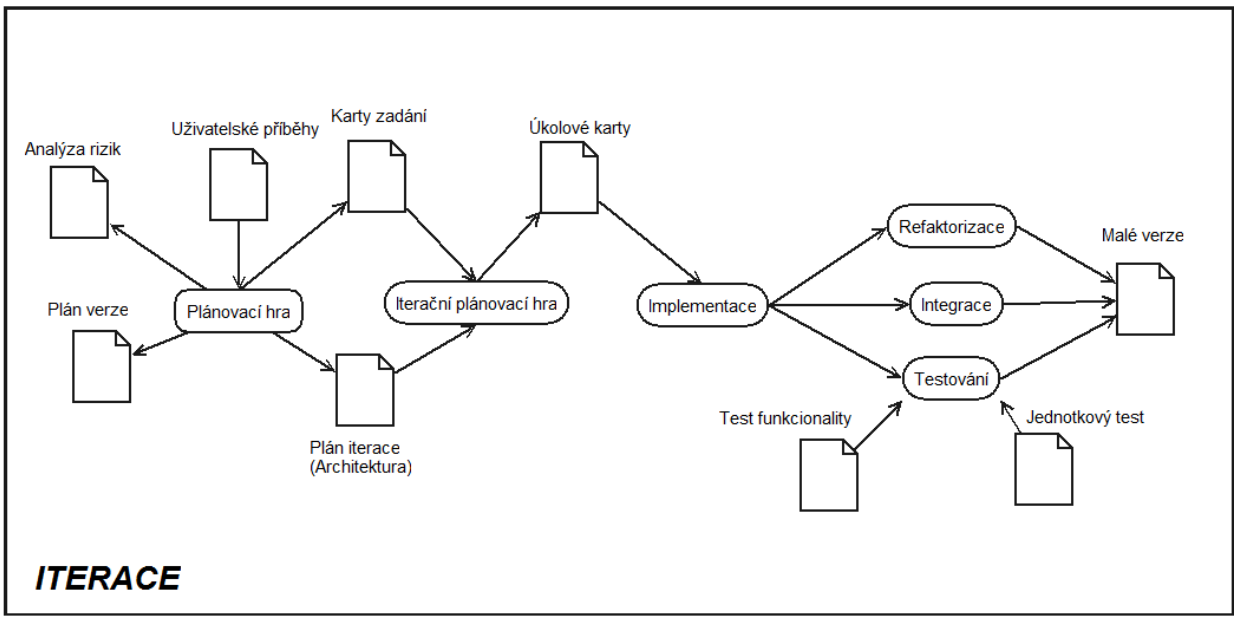
Metodika XP dělí vývojový proces do *iterací* trvající 1-4 týdny. V rámci každé iterace proběhnou všechny pracovní postupy, které metodika definuje.



Obrázek 10: Časové dělení vývojového procesu UP



Obrázek 11: Časové dělení vývojového procesu Scrum



Obrázek 12: Časové dělení vývojového procesu XP

Z obrázků 10, 11, 12 je patrné dělení jednotlivých metodik. UP je členěna na jednotlivé fáze, ve kterých probíhají samotné iterace. V metodice Scrum je to podobné. Na rozdíl od dvou předešlých, metodika XP se nedělí na jednotlivé fáze, ale v každé iteraci proběhnou všechny pracovní postupy, které definuje.

Fáze zahájení metodiky UP je velice podobná fázi předehry metodiky Scrum. Výstup obou fázi je specifikace požadavků (u metodiky UP se jedná o Případ užití, u metodiky Scrum o Product backlog). Odpovídajícím výstupem v metodice XP jsou Uživatelské příběhy.

Další činnosti probíhající v druhé fázi metodiky UP (fáze rozpracování) jsou v metodice Scrum zastoupeny v její první fázi, tedy ve fázi předehry. Patří zde především návrh architektury, plán reailizace a analýza rizik. Tyto tři činnosti jsou stejně tak zastoupeny i v metodice XP.

Následně se vývoj softwarového produktu přesouvá k samotné implementaci. Opět je viditelná podobnost mezi fázi tvorby metodiky UP a fázi hry metodiky Scrum. Ve všech třech metodikách probíhají stejné činnosti. Jedná se o implementaci, testování a integraci.

Poté, co je hotova další část produktu, může být výsledek představen zákazníkovi. V metodice Scrum se této předváděné části říká Demo a odpovídá malým verzím metodiky XP.

Ve všech třech metodikách probíhá také koncové zhodnocení výsledku iterace. V metodice UP tomu odpovídá pojem koncová revize, v metodice Scrum probíhá hodnotící a retrospektivní schůzka a metodika XP označuje toto pojmem refaktorizace a zpětná vazba zákazníka.

V další části této kapitoly jsou podrobněji popsány jednotlivé činnosti iterací.

6.3 Průběh a pracovní postupy ve vývojovém procesu

Na začátku každého vývojového procesu by se mělo stanovit, co je cílem procesu, jaké zdroje a kolik času je k dispozici a také shromáždit veškeré požadavky, které jsou kladeny na výsledný produkt.

Poté se již může začít s plánováním vývoje, kde jsou shrnuty veškeré informace, které mohou jakýmkoliv způsobem ovlivnit průběh vývoje.

Dále následuje analýza budoucího produktu. Požadavky, které jsou na něj kladeny se například rozdělí do kategorií nebo podle priority, s jakou je nutné je implementovat. Probíhají analýzy rizik a také se předběžně odhaduje čas, potřebný k implementaci první verze softwarového produktu. Následně se vývojový proces přesouvá do fáze návrhu, při němž se navrhuje a vybírá ta nejvhodnější architektura, volí se ty nejefektivnější postupy k implementaci.

Po této fázi se již vývojový proces dostává do stádia, kdy probíhá samotná implementace budoucího softwarového produktu.

Jakmile je hotová určitá část produktu, začíná její testování. Pokud validace proběhne úspěšně, nebrání nic tomu, aby byla hotová část integrována k ostatním. Poté, co jsou integrovány veškeré části produktu, je výsledek otestován jako celek.

Pokud vše dopadlo podle představ zákazníka, nebrání nic tomu, aby byl výsledek předán, tzn. nainstalován na počítače zákazníka a ten ho mohl začít bez problémů využívat.

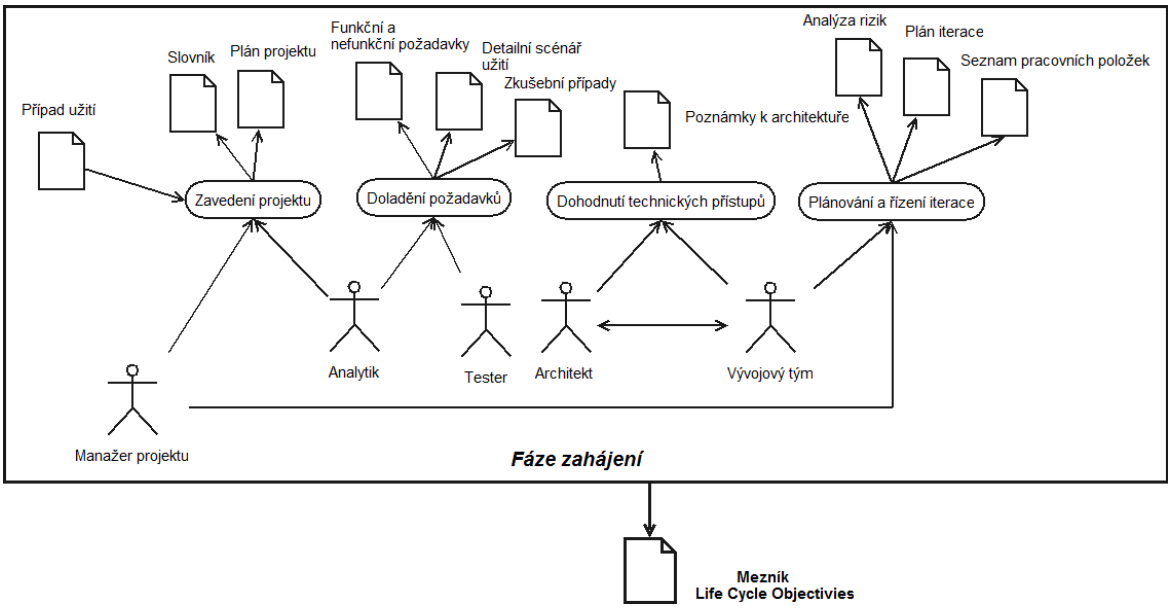
Průběh vývoje metodikou UP je následující: vývoj softwarového produktu začíná první fází, což je *fáze zahájení*, během ní probíhá *zavedení projektu*. V tomto kroku má důležitou roli *Analytik*, neboť do procesu vstupuje s představou zákazníka o produktu. Výstupem je slovník, zachycující všechny pojmy, které se mohou v průběhu vývoje vyskytnout. Další nepostradatelnou rolí v tomto kroku je *Manažer projektu*, který přichází s plánem, jak by měl produktu vypadat. Výstupem je hotový *plán projektu*, který je ústředním dokumentem, ve kterém mohou všichni členové týmu nalézt informaci o tom, jak se má vyvíjený produkt chovat, obsahuje popis mezníku poslední fáze a časové rozložení iterací v jednotlivých fázích. [4]

Poté následuje *rozpoznání a doladění požadavků*, během nichž se specifikují, analyzují a schvalují podmnožiny požadavků, které mají nejvyšší prioritu. Ty se budou implementovat a testovat nejdříve. V tomto kroku má důležité úlohy *Analytik* a *Tester*. Analytik má za úkol *identifikovat a nastínit požadavky*, tzn. zachytit funkční a nefunkční požadavky kladené zákazníkem na budoucí softwarový produkt, jako výstup vytvořit slovník a seskupit požadavky na další rozšíření a případ užití. Dále je jeho úkolem vytvořit *detailní scénáře případu užití*, což znamená popsat případ užití tak detailně, aby z něj bylo možné porozumět požadavkům, které budou schváleny zákazníkem. Vstupem je tedy diagram případu užití a výstupem detailní scénář. V neposlední řadě by také měl popsat aspoň jeden, nebo i více *detailů požadavků*, kterými by bylo možné produkt dále rozšířit. Hlavním úkolem Testera je vytvoření *zkušebních případů*, kterými bude ověřeno správné porozumění podmínkám specifikace. [4]

Zároveň s rozpoznáváním a doladěním požadavků probíhá *dohodnutí technických přístupů*. Cílem této aktivity je definice technických požadavků na systém tak, aby byly

splněny požadavky. *Architekt* by měl spolupracovat s vývojovým týmem na vytvoření počátečního náčrtu technických přístupů (architektury) navrhovaného produktu, zabezpečit, zda jsou technické rozhodnutí přiměřené a zda má tým dostatek informací k tomu, aby rozuměl přístupům, které bude používat. Výstupem této aktivity jsou *poznámky k architektuře (architecture notebook)*. [4]

Souběžně s předešlými třemi aktivitami probíhá *plánování a řízení iterace*, ve které se provádí analýza rizik a stanovuje se cíl iterace. Manažer projektu, zákazník a členové týmu souhlasí s tím, co se předpokládá, že bude během iterace implementováno. Výstupem této aktivity je plán iterace a *seznam pracovních položek*, což je soupis plánovaných prací, které je třeba provést v rámci projektu. Mezníku *Life Cycle Objectivies* a stanovení rozsahu projektu je dosaženo po dokončení všech těchto aktivit.



Obrázek 13: První fáze metodiky UP

Druhá je *fáze rozpracování*. Během této fáze probíhá mnoho aktivit paralelně a některé probíhají jak v první fázi zahájení, tak i v této druhé. Jsou to aktivity *rozpoznání a doladění požadavků a plánování a řízení iterace*. Navíc zde probíhají další čtyři aktivity.

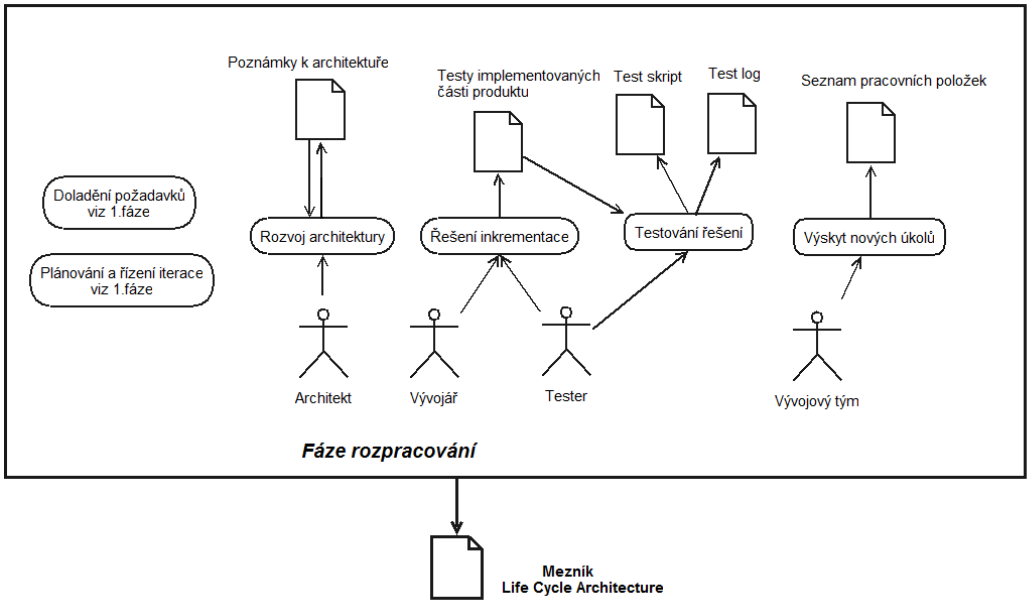
První z nich je *rozvoj architektury*. Architekt na základě poznámek k architektuře dotváří její podobu, kterou opět zaznamená.

Další aktivitou je *řešení inkrementace*, během ní Vývojář rozpoznává chování, relace a data produktu nezbytné pro implementaci určité funkcionality. Na základě technické specifikace a architektury navrhne design produktu a na základě softwarové implementace vytvoří testy, kterými ověří validitu konkrétní implementované části. Poté dále pokračuje na implementaci zdrojového kódu nové funkcionality nebo odstraňuje závady. Pokud všechny testy skončí úspěchem, může integrovat všechny provedené změny k ostatnímu kódu.

Následně Tester provádí *testování řešení*, tzn. testy implementuje a spouští. Výstupem je *Test skript a Test log*, ten zachycuje a analyzuje výsledky. Tyto informace jsou poté předávány ostatním členům týmu.

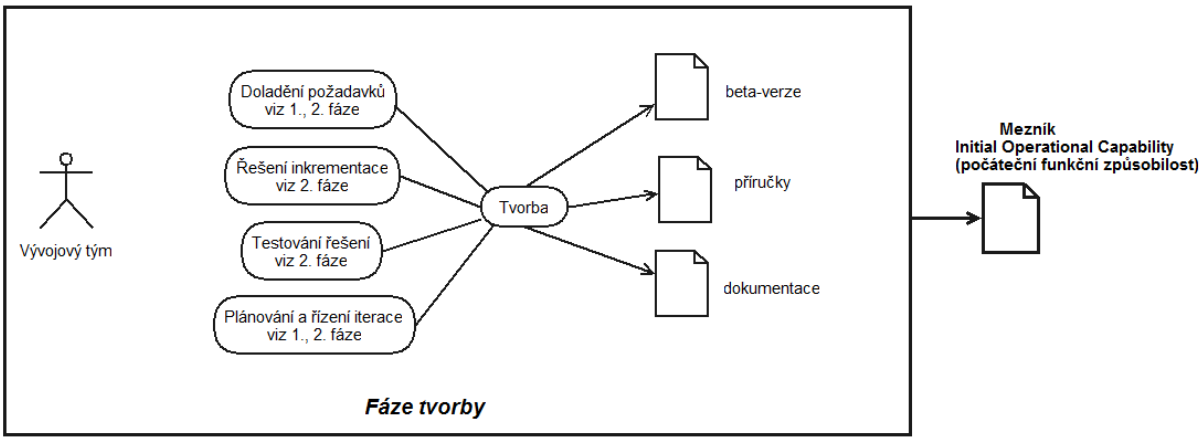
Nedílnou součástí je také *výskyt nových úkolů*. Pokud některý člen týmu zachytí změnu v požadavcích, je jeho úkolem ji zaznamenat do *seznamu pracovních položek*. [4]

Po úspěšném skončení aktivit fáze rozpracování, tzn. po dosažení mezníku *Life Cycle Architecture*, je vytvořen robustní a stabilní architektonický základ a tak je možné se posunout do další fáze.



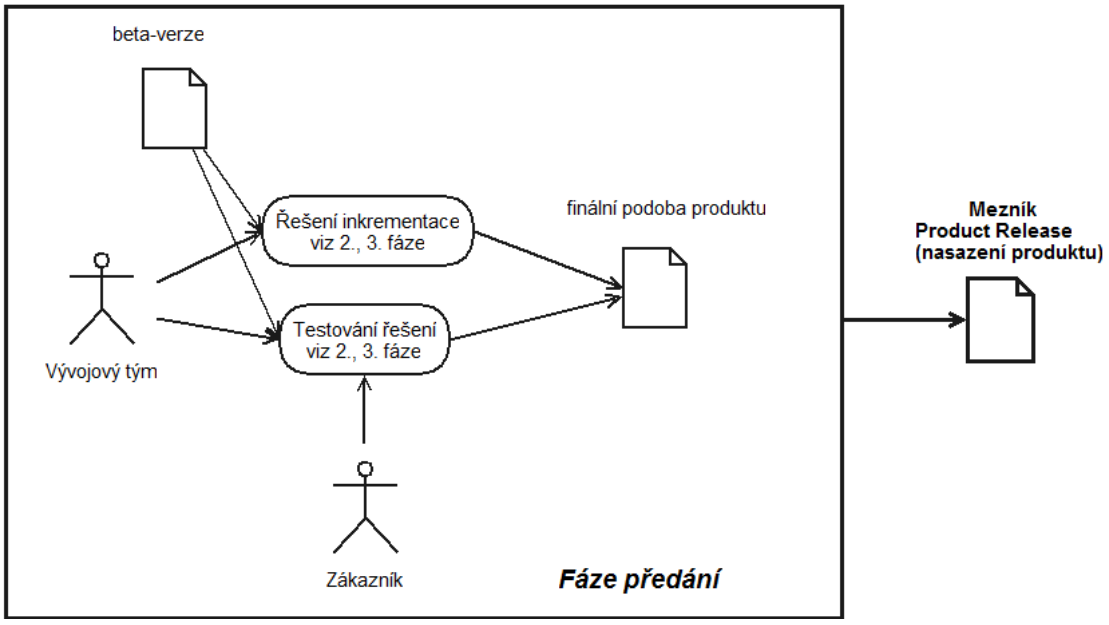
Obrázek 14: Druhá fáze metodiky UP

Třetí *fáze tvorby* obsahuje některé z aktivit, které již byly popsány dříve. Jedná se o *rozpoznání a doladění požadavků, řešení inkrementace, testování řešení, plánování a řízení iterace*, zaznamenávají se *nové úkoly*. Aby mohlo být dosaženo mezníku *Initial Operational Capability (počáteční funkční způsobilost)*, je nezbytné mít vytvořenou tzv. beta-verzi, což je stabilní produkt připravený k testování na počítačích uživatele. Dále se začínají tvořit uživatelské příručky a dokumentace. [4]



Obrázek 15: Třetí fáze metodiky UP

V poslední *fázi předání* se opět vyskytují aktivity, které již byly popsány. Aby byly beta testy s úspěchem dokončeny, probíhá opět *testování řešení, řešení inkrementace* a pokud se vyskytnou nové úkoly, tak i ty musí být zpracovány. Dále je nezbytné, aby zákazník souhlasil s výsledkem a popřípadě byla dohodnuta podpora produktu do budoucna. Splněním těchto podmínek je dosaženo mezníku *Nasazení produktu*.

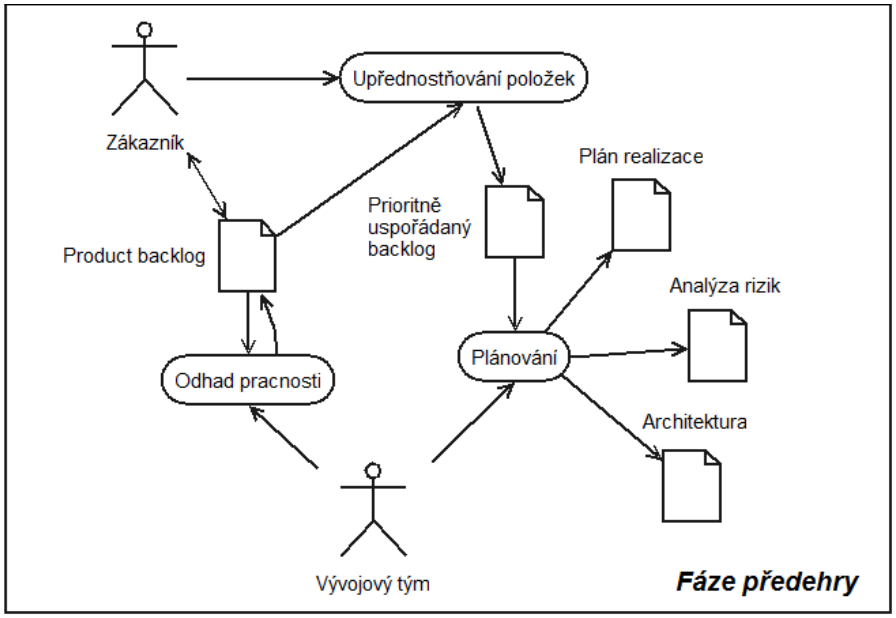


Obrázek 16: Čtvrtá fáze metodiky UP

Vývoj metodikou Scrum si můžeme představit jako hru, která se člení na fázi předehry, fázi hry a fázi dohry.

Na úplném začátku *fáze předehry* je sestaven tzv. *Product backlog*, což je specifikace požadavků kladených na vyvíjený produkt. Jak už bylo zmíněno výše, zákazník se v něm pokouší zachytit veškeré vlastnosti a funkce, které požaduje od budoucího produktu. Poté, co je sestaven backlog, začíná *odhad pracnosti (Effort Estimate)*, tzn. vývojový tým odhadne velikost a náročnost na realizaci každé položky backlogu a zákazník se na základě tohoto odhadu rozhodne, které položky jsou pro něj prioritní a které se naopak mohou odložit na pozdější realizaci. Jedná se o tzv. *upřednostňování* položek backlogu. Odhady jsou na začátku relativní, ale po pár sprintech dokáže tým přesně říci, jak dlouho jim bude trvat dodání dané funkcionality. [5]

Po zhotovení již prioritně uspořádaného backlogu může začít *plánování*, během něhož vývojový tým sestaví plán realizace. Nejedná se však o podrobný plán, jelikož na začátku vývoje nejsou známy všechny informace. Plán však obsahuje vše potřebné proto, aby se mohlo začít s vývojem softwarového produktu. Obsahuje například informace o tom, kolik sprintů bude potřeba, jak dlouho budou trvat, kolik členů nebo týmů se bude podílet na vývoji, datum předání hotové části produktu a podobně. Klíčovými vstupy pro plánování je prioritně uspořádaný backlog, odhadovaná rychlost vývoje, zákazníkem stanovený termín dokončení a nasazení produktu. Poté se navrhne architektura produktu a provádí se analýzy.



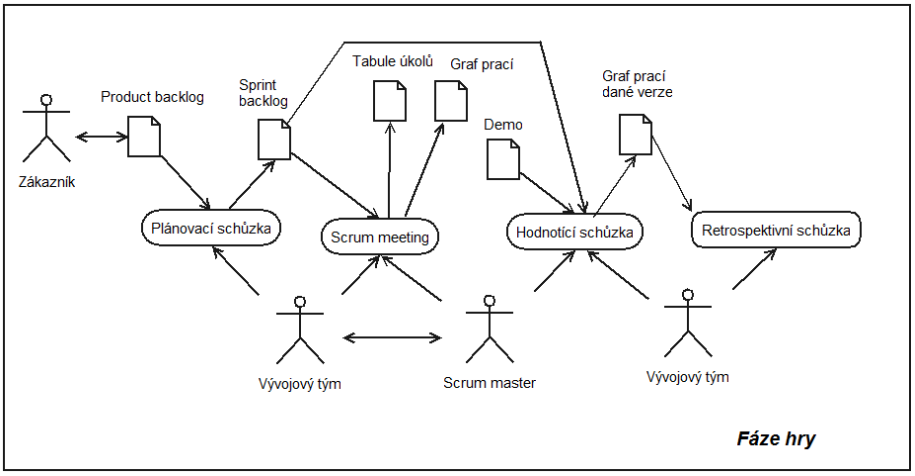
Obrázek 17: První fáze metodiky Scrum

Po dokončení těchto aktivit se vývoj přesouvá do *fáze hry*, během které probíhají samotné sprinty. Na začátku každého sprintu proběhne *plánovací schůzka (Sprint Planning Meeting)*, kde zákazník popíše vývojové týmu, která funkce či vlastnosti produktu jsou pro ně klíčové. Členové týmu poté od zákazníka zjistí všechny potřebné detaily a vyberou položky *Product backlogu*, které se toho týkají a ty přesunou do tzv. *Sprint backlogu*. Ten tedy zachycuje všechny funkce a úkoly, které budou implementovány během nadcházejícího sprintu. Ostatní funkce, které ještě nebyly implementovány se řeší na dalších plánovacích schůzkách během dalších sprintů. [5]

Tím tedy může začít samotný vývoj produktu. Každý den sprintu začíná *Scrum meetingem*, na kterém se sejdou všichni členové vývojového týmu a Scrum master. Během této schůzky sdělí všichni ostatním členům, co zvládli udělat včera a co se chystají splnit do dalšího setkání. Tím má celý tým přehled o tom, co již je splněno a co ještě zbývá dodělat do konce sprintu. Scrum master nemá za úkol zaznamenávat, kdo se opozdil s implementací nebo podobně, ale je jeho úkolem dělat přehled o tom, které části a funkcionality produktu jsou již hotovy, aby i zákazník byl „v obraze“. Tento přehled se zapisuje na *tabuli úkolů (Task board)* a jsou na ní viditelné veškeré úkoly sprintu rozdělené do kategorií podle toho, které chybí udělat, které se právě implementují, které je třeba ještě otestovat a které jsou již hotovy. Scrum master dále zachycuje množství práce, které je ještě nutné dodělat během sprintu v tzv. *grafu prací (Burndown Chart)*. [5]

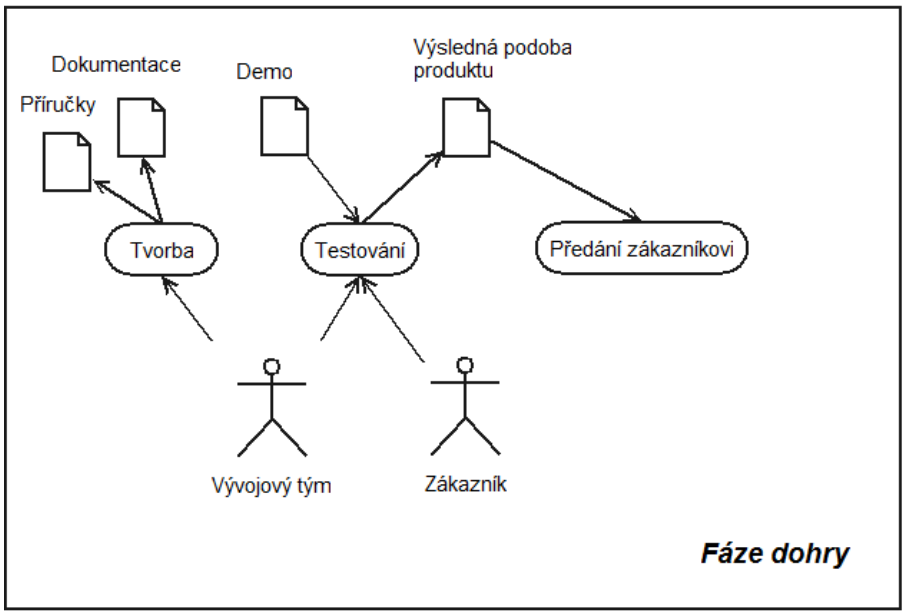
Každý sprint končí *hodnotící schůzkou (Sprint Review Meeting)*, kde tým demonstruje ostatním zainteresovaným stranám tu část produktu, která byla během něj zhotovena. Každé takovéto části se říká *Demo*. Dále se vyhodnocuje míra úspěšnosti dokončení po-

ložek sprint backlogu a celkového cíle sprintu. To vše se zaznamená do *grafu prací aktuální verze (release burndown chart)* a může se tak porovnat, zda bylo dosaženo všeho, co bylo naplánováno. Z toho obvykle vyplyne diskuze, zda neexistuje něco, co by se dalo ještě dále zlepšovat, aby byl výsledek lepší, a v čem by měl pokračovat další sprint. Jedná se o tzv. *retrospektivní schůzku (Sprint Retrospective Meeting)*. [5]



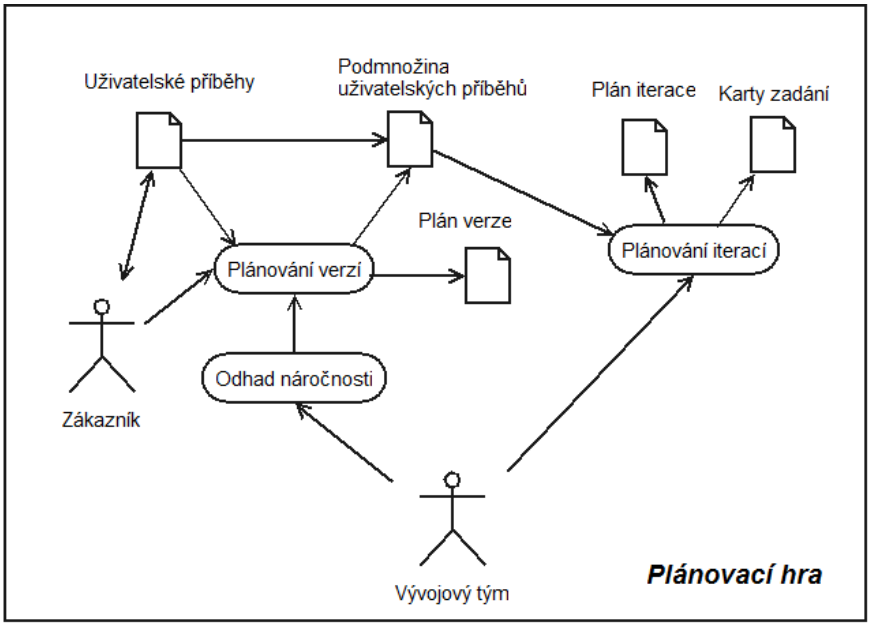
Obrázek 18: Druhá fáze metodiky Scrum

Poté, co produkt splňuje již všechny požadavky zákazníka, tzn. jsou splněny všechny položky product backlogu, začíná *fáze dohry*. Během této fáze se již neimplementují žádné změny, ale jsou prováděny testy produktu jako celku, píše se dokumentace a manuály, a produkt se chystá k finálnímu předání a nasazení na počítačích zákazníka.



Obrázek 19: Třetí fáze metodiky Scrum

Jedním ze základních pracovních postupů, zastupující klasické plánování v Extrémním programování, je *Plánovací hra*. Metodika XP dělí celé plánování produktu na menší celky, jako je plánování verzí, trvající přibližně 2 až 3 měsíce. To je děleno na plánování iterace, trvající 2 až 3 týdny. Plán iterace je ještě dělen na naplánování jednotlivých úkolů, které jsou plněny obvykle během 1 až 2 dnů. Plánovací hra se skládá ze dvou hlavních aktivit, a to *plánování verzí* a *plánování iterací*. V první výše zmíněné představuje zákazník své uživatelské příběhy vývojovému týmu, programátoři odhadnou jejich náročnost a zákazník z nich poté vybere množinu, která se bude realizovat v přísti verzi produktu. Při plánování iterací zákazník opět vybírá uživatelské příběhy, obvykle jsou to ty z plánování verzí, avšak nyní musí vybrat ty klíčové, které budou implementovány během iterace. Programátoři je poté rozloží do jednotlivých úkolů a každý z týmu si vybere ty, na kterých bude pracovat. Provede se odhad, zda jsou schopni je včas naimplementovat. Pokud se zjistí, že jich je příliš mnoho a nestihly by se včas naimplementovat, je na zákazníkovi, aby vybral ty méně důležité, které budou řešeny až v příští iteraci. Výhodou Plánovací hry je to, že poskytuje rychlou zpětnou vazbu a dlouhodobý plán (*plán verze*), ze kterého plynou krátkodobější cíle (*plán iterace*), které vedou tým k úspěšnému dokončení vývoje celého produktu. [6]



Obrázek 20: Plánování v metodice XP

Metafora je způsob, jak po analýze definovat produkt a vztahy mezi jeho částmi běžně používaným slovník tak, aby tomu porozuměli jak členové vývojového týmu, tak i zákazník. Metafora není po dobu vývoje produktu statická, ale s tím jak se produkt vyvíjí, vyvíjí se i samotná metafora.

Strategie pracovního postupu návrhu je vytvořit *jednoduchý design*, který bude splňovat podmínky a požadavky kladené na produkt a má tyto vlastnosti: splňuje všechny testy, neobsahuje duplicitní kód, kód je snadno srozumitelný programátorům, obsahuje co nejméně tříd a metod, přitom je však stále zachována jeho funkcionalita. Výhodami jednoduchého designu je především jeho flexibilita a svižnost, neboť změny, které je nutné provést se vždy uskuteční lépe a rychleji než u velkých a složitých návrhů.

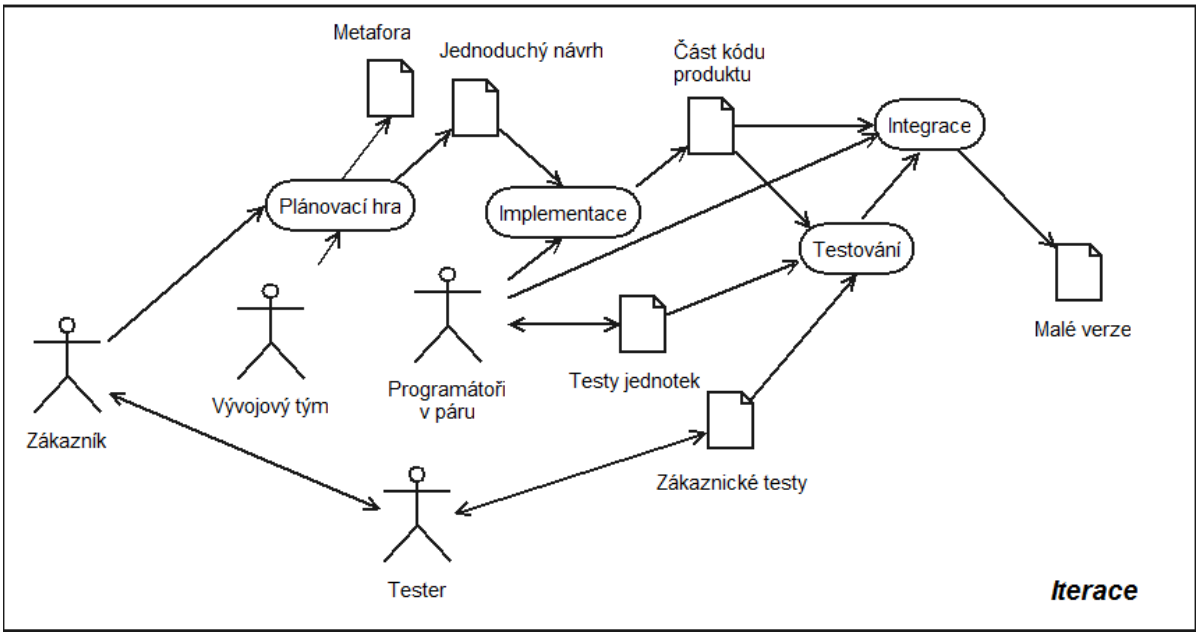
Implementace probíhá v metodice XP specificky a to v páru. Jedná se o tzv. *párové programování*, kdy dva programátoři tvořící pár sdílejí jeden počítač s jedním monitorem, myší i jednou klávesnicí. Tím je zajištěna neustálá kontrola druhým v páru, že daný způsob řešení je ten nejefektivnější a tím i nejpřijatelnější. Díky tomu je lepší design, testy i samotný zdrojový kód. Párové programování také vede ke zlepšení znalostí a komunikace v týmu, jelikož páry se mění a tím si programátoři předávají znalosti a zkušenosti. Z toho plyne další specifický rys metody XP a to *společné vlastnictví kódu*. V praxi to vypadá tak, že pokud kterýkoliv z členů týmu má nápad, jak změnit kód k lepšímu, nebrání mu nic v tom, aby ho změnil. Musí však dbát na to, aby byla zachována funkcionalita a všechny testy i po úpravě byly úspěšně provedeny. [6]

Celý životní cyklus produktu je vedený *vývojem řízeným testy*. Tím je myšleno to, že jakékoliv změně ve funkcionalitě či vlastnostech produktu předchází test, který ověří správnost doposud implementovaných části. Teprve po provedení dané změny násle-

duje opět test, který nás ujistí o správnosti provedených změn. Dalším druhem testů jsou *zákaznické testy*. Zákazník si stanoví, co má být v konkrétním uživatelském příběhu otestováno a Tester mu pomůže z jeho sestavením. Pak je již na zákazníkovi, kdy si ověří funkčnost produktu, neboť jeho testy jsou automatizované, díky tomu se mohou spouštět kdykoliv. Zákazník má tak možnost vidět, jak se produkt posouvá ve svém vývoji. [6]

Jedním z cílů XP je zajistit, aby zákazník měl neustály přehled o pokroku ve vývoji softwarového produktu. Z toho plyne nutnost často, někdy i několikrát denně, provádět integraci nových části kódu. Tato *neustálá integrace* probíhá okamžitě, jakmile některý z párů programátorů má hotovou tu část produktu, na které pracovali. Integrace probíhá opět za pomoci testů. Před integrací si pár ověří funkčnost doposud hotových částí, integrují svou část kódu a následně opět spustí testy, které musí mít 100% úspěšnost. Pokud nemají, je jejich povinností provést takové změny a úpravy, aby této úspěšnosti dosáhli. Výhodou tohoto přístupu k integraci je snížení konfliktů, které by mohly nastat při jednorázové integraci velkého kusu kódu a zajišťuje to, že všichni členové týmu pracují s nejnovější verzí kódu produktu. [6]

Tím, že se v metodice XP téměř neustále uvolňují a předávají nové tzv. *Malé verze*, má zákazník neustálý přehled o produktu a jeho funkčnosti a vývojový tým tak dostává rychle zpětnou vazbu, na kterou může například reagovat další malou verzí produktu. Výhodou je také to, že se v brzké době zjistí, zda funkce a vlastnosti produktu, které se vytváří, jsou těmi vlastnostmi, které zákazník od produktu požaduje. [6]



Obrázek 21: Iterace metodiky XP

Na obrázcích 13, 17 a 20 je vidět, čím celý vývojový proces v jednotlivých metodikách začíná. V metodice UP je to *fáze zahájení*, v metodice Scrum se jedná o *fázi předehry* a metodika XP označuje tuto fázi jako *plánovací hru*.

V každé z těchto fází probíhá stejná činnost a to specifikace požadavků, i když každá metodika toto označuje svým pojmem. V UP to jsou *případy užití*, ve Scrum metodice se jedná o *Product backlog* a v metodice XP jsou to *uživatelské příběhy*.

Další činnost, kterou můžeme nalézt ve všech metodikách, je odhad rizika a naplánování dané iterace. Pojem *analýza rizik* používají všechny tři metodiky, avšak pro plán iterace používá každá svůj. V XP se jedná přímo o pojem *plán iterace*, metodika Scrum však používá *plán realizace* a v metodice UP bychom našli pojem *plán projektu*, ze kterého poté vzniká *plán iterace*.

Nedílnou součástí první části vývoje všech tří metodik je také návrh samotné *architektury*. Tomu předchází *odhad pracnosti* (pojem metodiky UP a Scrum) či *náročnosti*, jak je to označováno v metodice XP.

Pokud vývojový proces prošel všemi těmito činnostmi, není důvod, aby nepokračoval dál.

Další části vývoje je samotná implementace softowarového produktu. Metodika UP tuto část dělí ještě na dvě fáze (viz obrázek 14 a 15). Jedná se o *fáze rozpracování* a *tvorby*. Metodika Scrum implementaci řeší v rámci *fáze hry* (viz obrázek 18). Vývoj metodikou XP již není dělen do fázi, ale přímo probíhá *implementace*.

Po zhotovení určité části produktu, která se opět v každé metodice nazývá jinak (UP - *beta-verze*, Scrum - *demo*, XP - *malé verze*) začíná samotné testování. Průběh testování je ve všech metodikách přibližně stejný. Pokud nejsou testy splněny se 100 % úspěšností, není možné danou část produktu integrovat k ostatním.

Na konci vývojového cyklu v každé metodice (viz obrázek 16, 19 a 21) proběhne zhodnocení průběhu a představení implementované části. Metodika Scrum má pro tuto činnost přímo pojem *hodnotící* a *retrospektivní schůzka*, které se účástní všechny zainteresované strany a zákazníkovi je představeno funkční demo. Stejnou činnost si můžeme představit i v metodikách UP a XP.

Pokud výsledek odpovídá požadavkům zákazníka, zbývá už jen nainstalovat daný produkt na zákazníkův počítač.

V metodice UP se to děje ve *fázi předání* (viz obrázek 16). Produkt se naposledy otestuju už jako funkční celek a předá se zákazníkovi. Společně s produktem se předávají i příručky a dokumentace, vytvořené ve fázi tvorby (viz obrázk 15).

Průběh předání je v metodice Scrum obdobný. Samotné nasazení produktu se děje ve *fázi dohry* (viz obrázek 19). Zde proběhne finální otestování produktu a pokud vše dopadne podle očekávání, nebrání nic tomu, aby byl produkt předán zákazníkovi. V této fázi se také tvoří dokumentace a příručky.

Metodika XP řeší průběh předání a nasazení produktu u zákazníka velice podobně jako dvě předešlé metodiky. Také probíhá finální otestování produktu a to jak vývojový týmem, tak i samotným zákazníkem (viz obrázek 21). Po úspěšném skončení testů, je produkt předán zákazníkovi, který je aktivně začne využívat na svých počítačích.

7 Společný slovník pojmů

7.1 Přehled pojmů

UP	Scrum	XP
Scénář případu užití	Product backlog	Uživatelské příběhy, Karty zadání
Seznam pracovních položek	Sprint backlog	Úkolové karty
Analýza rizik	Analýza rizik	Riziko
Beta-verze, Baseline	Demo	Malé verze
Plánování	Plánovací schůzka sprintu	Plánovací hra
Iterace	Sprint	Iterace
Inkrementace	Každodenní schůzka	Iterační plánovací hra
Fáze zahájení	Fáze předehry	Iterace
Fáze rozpracování		
Fáze tvorby	Fáze hry	
Fáze předání	Fáze dohry	
Analytik	Scrum master	Tester
Architekt		Kouč
Manažer projektu	Manažer projektu	Manažer projektu, Velký šéf
Vývojář	Vývojový tým	Programátoři v páru
Tester	Tester	Programátoři v páru
Zákazník	Chickens	Zákazník na pracovišti
Manažer projektu	Scrum master	Stopař
Tester, vývojář	Scrum master, Tester, Vývojář	Konzultant
Vedení	Pigs	Vedení
Vývojový tým	Pigs	Vývojový tým
Architektura	Architektura	Architektura
Artefakt	Sprint backlog, Demo, …	Karty zadání, Úkolové karty, …
Plán projektu	Graf prací	Metriky
Fáze předání	Retrospektivní schůzka	Refaktorizace
Fáze předání	Hodnotící schůzka	Testy
Slovník	Product backlog	Metafora
Plán projektu	Plán realizace	Plán verze, plán iterace

Obrázek 22: Přehled pojmů

7.2 Vysvětlení jednotlivých pojmů

Scénář případu užití: představuje konkrétní případ v metodice UP, který odpovídá skutečnému případu v reálném světě na základě vstupů od aktéra. Každý scénář je popisem toho, jak se produkt bude chovat v různých situacích, tzn. na základě různých vstupů.

Product backlog: jedná se o základní nosič informace metodiky Scrum vyvíjeného produktu. Může mít formu tabulky, či uživatelských příběhů apod., kde jsou zaznamenány všechny funkce a vlastnosti, které je nutné implementovat. Backlog smí modifikovat pouze manažer projektu, ale ke čtení je přístupný všem zainteresovaným ve vývoji.

Uživatelské příběhy a z nich vzniklé *Karty Zadání*: specifikace požadavků zákazníka a nosiče zadání pro vývoj softwarového produktu metodikou XP.

Seznam pracovních položek: soupis plánovaných prací, které je třeba provést v rámci vývoje metodikou UP.

Sprint backlog: seznam prací a úkolů, které je nutné dokončit během sprintu metodiky Scrum. Jedná se o podmnožinu Product backlogu, kterou si však vytváří vývojový tým.

Úkolové karty: konkrétní popis úkolu a k implementaci jednotlivých funkcí vyvíjeného softwarového produktu metodikou XP.

Analýza rizik: jejím úkolem je zjistit všechny možné okolnosti, které by mohly narušit plynulý průběh vývoje a dále připravit plán, jak v takovém případě reagovat a postupovat.

Riziko: potenciální událost nebo situace, která může nastat a ovlivnit tak úspěšné dokončení vývoje softwarového produktu.

Baseline (základní linie): interní (externí) množina artefaktů vytvořených příslušnou iterací.

Beta-verze: stabilní verze vyvíjeného produktu, kterou je nezbytné otestovat.

Demo: funkční verze vyvíjeného softwarového produktu, která je na konci každého sprintu metodiky Scrum předvedena zákazníkovi.

Malé verze: implementována a integrována část vyvíjeného produktu, zachycující aktuální požadavky zákazníka v metodice XP.

Plánování: vývoj metodikou XP začíná tímto pracovním postupem, během něhož probíhá stanovení požadavků a náčrt architektury budoucího softwarového produktu.

Plánovací schůzka sprintu (Sprint Planning Meeting): probíhá na začátku každého sprintu metodiky Scrum. Zákazník vybírá z Product backlogu funkce podle priority a na základě těchto informací sestavuje vývojový tým Sprint backlog, který budou plnit v nadcházejícím sprintu.

Plánovací hra: jedná se o plánovací proces Extrémního programování. Účastníci jsou všichni členové týmu a vedení. Úkolem vedení a zákazníků je specifikovat, co má daný produkt dělat. Vývojový tým určí, kolik každá část bude stát a termín dokončení. Cílem je vytvoření plánu vyvíjeného softwarového produktu a karet zadání.

Iterace: představuje samostatný problém, funkcionalitu vyvíjeného produktu jako část, kterou lze řešit nezávisle na ostatních. Skládáním (řešením) iterací jednu za druhou se vytváří konečná podoba vyvíjeného softwarového produktu metodikou UP.

Sprint: je základní vývojovou iterací metodiky Scrum. Trvá obvykle mezi 2 a 4 týdny, většinou 30 dní. Konkrétní náplň každého sprintu je určována v průběhu každodenní schůzky – Scrum meetingu.

Iterace: období jednoho až čtyř týdnů vývoje metodikou XP. Na začátku si zákazník stanoví funkce, kladené na produkt, které mají být v dané iteraci implementovány. Na konci každé iterace jsou spouštěny testy funkcionality zákazníka, čímž se ověří funkčnost a úspěšnost dané iterace.

Inkrementace (přírůstek): je rozdíl mezi dvěma základními liniemi (Baseline) metodiky UP, které jsou generovány v každé iteraci. Inkrementace je dílčí krok směřující k finálnímu předání produktu.

Každodenní schůzka (Daily Scrum Meeting): setkání celého vývojového týmu, které se koná každý den, trvá přibližně 15-30 minut a je zde shrnuta veškerá práce, která byla vykonána, a naplánována práce na příští den. Celou schůzku vede Scrum Master.

Iterační plánovací hra: velice podobná Plánovací. Účastníci jsou však pouze programátoři, kteří namísto karet zadání používají úkolové karty. Celá hra se dohraje během jedné iterace, to znamená do jednoho až čtyř týdnů. Jednotlivé fáze a tahy hry jsou podobné těm v Plánovací hře.

Fáze zahájení (inception): první fáze metodiky UP, vývoj projektu startuje, milníkem jsou záměry životního cyklu (Life Cycle Objectives).

Fáze předehry: vývoj softwarového produktu metodikou Scrum je na samém začátku, začíná se s plánováním a navrhuje se architektura a design budoucího produktu.

Fáze rozpracování (elaboration): druhá fáze metodiky UP, vzniká architektura produktu, milníkem je architektura životního cyklu programového díla (Life Cycle Architecture)

Fáze tvorby (construction): třetí fáze metodiky UP, je vytvářen software produktu, milník je počáteční funkční verze (Initial Operational Capability).

Fáze hry: v této fázi probíhá samotný vývoj softwarového produktu metodikou Scrum, tzn. sprinty, v rámci nichž probíhá psaní zdrojového kódu a následně revize a případné přizpůsobení produktu.

Fáze předání (transition): čtvrtá fáze metodiky UP, softwarový produkt je nasazován na pracoviště uživatele, milníkem je nasazení produktu (Product Release).

Fáze dohry: v poslední fázi metodiky Scrum probíhá samotné nasazení produktu u zákazníka, tzn. uzavření vývoje softwarového produktu.

Analytik: je osoba (nebo i osoby) ve vývojovém týmu UP, která zastupuje zákazníka a koncového uživatele. Jeho úkolem je shromáždit vstupní informace a požadavky, jaké

zákazník klade na vyvíjený softwarový produkt, a dále pro ně nastavit priority, aby bylo možné začít s plánováním celého projektu.

Tester: hlavním úkolem testera v metodice XP není provádění testů, ale pomoc zákazníkovi při psaní testů funkcionality.

Architekt (projektant): je zodpovědný za definici architektury softwaru vyvíjeného metodikou UP, úzce spolupracuje s projektovým manažerem při plánování projektu. Při ne příliš rozsáhlých projektech může být jedna osoba architektem a zároveň i projektovým manažerem.

Scrum master: osoba odpovědná za průběh vývoje metodikou Scrum, dbá na to, aby byly využívány všechny její výhody, komunikuje se zákazníkem a dělá průběžné hodnocení jednotlivých sprintů.

Kouč: je jedna ze dvou rolí představující pojem řízení XP. Předpokladem pro ideálního kouče je výborná schopnost komunikace nejen s vývojovým týmem a zákazníkem, ale i technická vybavenost (v jiných týmech většinou zastoupený jako vedoucí programátor, či systémový architekt). Koučování znamená být k dispozici jako vývojový partner, zejména pro nové a začínající programátory, při složitějších technických úkolech a obecně pomáhat vývojářům s jednotlivými technickými dovednostmi, dále dohlížet na dlouhodobé cíle refaktorizací a jejich dosažení, a v neposlední řadě přibližovat a objasňovat jednotlivé stránky procesu manažerům na vyšších úrovních.

Manažer projektu: vede plánování projektu vyvíjeného metodikou UP, spolupracuje se zákazníkem a jeho důležitým úkolem v týmu je zajistit, aby tým byl soustředěný na dokončení projektu. Zároveň je zodpovědný za analýzu rizik projektu.

Manažer projektu: jediná osoba, která smí dělat úpravy v Product backlogu metodiky Scrum. Jejím úkolem je pozorně naslouchat vývojovému týmu a zákazníkovi a jejich požadavky a návrhy přenášet do backlogu.

Manažer projektu: role v týmu metodiky XP. Jeho úkolem je správně rozdělit prostředky, které jsou k dispozici.

Vývojář: má zodpovědnost za jednotlivé části vyvíjeného softwarového produktu metodikou UP. Dále také provádí integraci návrhu do celkové architektury, navrhuje prototypy, které poté implementuje. Stará se také o integraci komponent do větších funkčních bloků produktu.

Párové programování: speciální programovací technika v Extrémním programování, kdy dva programátoři sedí u jednoho počítače s jednou myší, jednou klávesnicí i jedním monitorem. Páry nejsou na stálo, ale v průběhu dne se mění.

Tester: se stará o veškeré aktivity související s pracovním postupem testování. To zahrnuje nalezení, definici, implementaci a řízení nezbytných testů, stejně tak jako zaznamenávání výsledků testů a jejich analýzu.

Zákazník (investor): reprezentuje všechny osoby, které jsou nějakým způsobem zainteresovány do vývoje metodikou UP, tzn. můžou jistým způsobem ovlivnit výslednou podobu produktu, a nejsou členy vývojového týmu.

Chickens: jedna ze skupin rolí, které jsou zastoupeny v metodice Scrum. Patří zde všichni, kteří nepatří do vývojového týmu.

Zákazník na pracovišti: metodika XP požaduje přítomnost zákazníka, který bude využívat budoucí produkt při vývoji, aby byla neustále udržována zpětná vazba a vývojový tým měl kdykoliv možnost doplnit si informace o požadavcích zákazníka kladených na vyvíjený produkt.

Stopař: jeho úkol je především sledovat všechny metriky, které jsou v dané části vývoje důležité a informovat tým o tom, jak se liší či neliší skutečnost od předpokládaného. Současně je jeho úkolem i provozování Plánovací hry, z toho plyne, že stopař musí perfektně znát pravidla hry a být kdykoliv připraven je prosazovat. Role stopaře a kouče v metodice XP může být obsazena jednou a toutéž osobou.

Konzultant: jelikož se v metodici XP zabývají všichni programátoři všemi částmi kódu, to znamená nikdo není expert na žádnou konkrétní oblast, je občas nutné poradit se o daném problému s technickým konzultantem.

Pigs: jedna ze skupin rolí, které jsou zastoupeny v metodice Scrum. Patří zde celý vývojový tým včetně vedení.

Architektura: popisuje náčrt vývoje softwarového produktu, často reprezentována jako množina architektonických pohledů, obsahuje také předpoklady, vysvětlení a důsledky rozhodnutí, která byla provedena při návrhu architektury.

Artefakt: jsou produkty práce, představují hmatatelné netriviální položky, které jsou vytvořené nebo dále používané v úkolech. Artefakt může být složen z dalších artefaktů a často je základem dále používaných prostředků.

U ostatních metodik se jedná například o vytvoření Sprint backlogu, Demo, Karet zadání, apod.

Plán projektu: důležitý dokument při vývoji metodikou UP. Jsou zde uvedeny informace o chování vyvíjeného produktu, mezníky jednotlivých fází a časové rozložení iterací.

Graf prací (Burndown chart): grafické znázornění práce, která ještě zbývá udělat reprezentována jako funkce zbývajícího času. Na ose Y je zbývající práce a na ose X čas.

Metriky: základní nástroj pro řízení XP. Jedná se například o poměr mezi časem, který se odhaduje pro vývoj, a kalendářním časem, který je základní mírou Plánovací hry, jež vývojovému týmu umožňuje nastavit tzv. rychlost projektu. Pokud máme méně kalendářního času na dané odhadované množství vývoje, to znamená poměr se zvyšuje, může to značit, že týmový proces vývoje funguje dobře. Může to ale také znamenat, že tým dělá, kromě plnění požadavků, jen malé procento z ostatní práce, kterou by měl zvládnout, což

se projeví až v následujícím období. Metriky jsou znázorňovány v grafu, jež má každý člen vývojového týmu na očích.

Retrospektivní schůzka (Sprint retrospective meeting): na konci každého sprintu se vývojový tým sejde a diskutuje o problémech, které se vyskytly a věcech, které fungovaly a naopak nefungovaly. Tím se jim dostává zpětné vazby a mají tak možnost provést změny do dalšího sprintu.

Refaktorizace: změna zdrojového kódu produktu v průběhu vývoje, která však neovlivní jeho chování, ale zlepšuje některé jeho jiné vlastnosti, například jednoduchost, pochopitelnost, výkonnost a podobně.

Hodnotící schůzka (Sprint Review Meeting): probíhá na konci sprintu a vývojový tým demonstruje zákazníkovi a dalším zainteresovaným stranám hotové položky Product backlogu, které implementovali v uplynulém sprintu.

Testy: neoddělitelná součást vývoje metodiky XP, při které se spouští zautomatizované sady podnětů, které však neovlivní stav vyvíjeného produktu, a čeká se na jeho správnou odezvu, kterou se zhodnotí úspěšnost vývoje dané části produktu. Existují dvě skupiny testů:

- *Test funkcionality*: napsaný z pohledu zákazníka. Spouští se obvykle po každé iteraci a nemusí vždy proběhnout na 100%, avšak při poslední iteraci je nutná 100% správnost.
- *Jednotkové testy*: napsaný z pohledu programátora. Spouští se před a po změně ve zdrojovém kódu a měl by vždy probíhat na 100% správně.

Slovník: zachycuje veškeré pojmy, které se mohou objevit v průběhu vývoje metodikou UP.

Metafora: jakýsi příběh, kterým si celý tým společně se zákazníkem sděluje, jak systém funguje běžně používanými slovy. Tím se odbourává komunikační bariéra mezi odborníky (vývojovým týmem) a zákazníkem a omezí se výskyt případů nepochopení požadavků.

Plán realizace: obsahuje informace nezbytné pro to, aby začal samotný vývoj produktu. Například na kolik sprintů je rozdělen vývoj, kolik členů týmu bude zapojeno, datum předání hotového produktu a podobně.

Plán verze: z uživatelských příběhů se na základě odhadu náročnosti vyberou ty, které budou implementovány v nadcházející verzi produktu.

Plán iterace: jsou vybrány uživatelské příběhy z plánu verze, které jsou poté implementovány v rámci dané iterace metodiky XP.

8 Závěr

Obsahem bakalářské práce bylo srovnání vybraných softwarových procesů a to Unified Software Development Process, Scrum Development Process a Extreme Programming.

První část byla věnována stručné teorii ze softwarového inženýrství, která měla zajistit, že je čtenář uveden do problematiky.

V dalších částech bakalářské práce byly popsány postupně všechny tři výše zmíněné metodiky. Bylo zde uvedeno časové dělení jednotlivých metodik, základní pracovní postupy, činnosti a role, které se vyskytují při vývoji softwarového produktu danou metodikou.

V části věnované samotnému srovnání byl chronologicky popsán průběh vybraných částí metodik. Byly zde uvedeny role, artefakty a činnosti, které vstupují do jednotlivých fází metodik a také jejich výstupy.

Poslední část bakalářské práce je věnována společnému slovníku pojmů, aby měl čtenář ucelený přehled o klíčových výrazech používaných v jednotlivých metodikách.

V problematice softwarových procesů je mnoho námětů k dalšímu výzkumu. Jedná se například o možnosti, jak vhodně skloubit metodiky, aby bylo možné eliminovat nevýhody jedné a využívat výhod druhé. Jedním z takových pokusů je skloubení metodiky XP a Scrum, na který jsem narazila na webových stránkách při hledání informačních zdrojů k mé práci. Dalším námět k budoucímu výzkumu je například přizpůsobování vlastností agilních metodik tak, aby je bylo možné využívat i ve větších vývojových týmech pro vývoj rozsáhlejších produktů, aby i velké softwarové giganty, sériově produkující software, mohly těžit z jejich výhod. S touto myšlenkou je však třeba nakládat promyšleně, aby zůstal viditelný rozdíl mezi tradičními (rigorózními) a agilními metodikami.

9 Reference

[1] BECK, K. *Extrémní programování: Knihovna programátora.* 1. vyd Praha: Grada, 2002. 158 s. ISBN 80-247-0300-9

[2] KADLEC, V. *Agilní programování - Metodiky efektivního vývoje softwaru.* 1. vyd Brno: Computer Press, 2004. 280 s. ISBN 80-251-0342-0

[3] ARLOW, J., NEUSTADT I. *UML a unifikovaný proces vývoje aplikací* 1. vyd Brno: Computer Press, 2003. 388 s. ISBN 80-7226-947-X

[4] IBM CORPORATION. *OpenUP* [online].
URL <http://epf.eclipse.org/wikis/openup/> [citováno 3. dubna 2010]

[5] MOUNTAIN GOAT SOFTWARE. *Scrum* [online].
URL <http://epf.eclipse.org/wikis/scrum/> [citováno 3. dubna 2010]

[6] IBM CORPORATION AND OBJECT MENTOR. *XP* [online].
URL <http://epf.eclipse.org/wikis/xp/> [citováno 3. dubna 2010]

[7] HORDĚJČUK, V. *Softwarové inženýrství* [online].
URL <http://voho.cz/wiki/softwarove-inzenyrstvi/> [citováno 20. února 2010]

[8] Agile Alliance. *Home Page* [online].
URL <http://www.agilealliance.org/> [citováno 20. února 2010]

[9] Extreme Programming. *Home Page* [online].
URL <http://www.xprogramming.com/> [citováno 20. února 2010]

[10] PALETA, P. *Co programátory ve škole neučí aneb Softwarové inženýrství v reálné praxi* 1. vyd Brno: Computer Press, 2003. ISBN 80-251-0073-1

[11] BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů - Kategorizace, agilní metodiky, vzory pro návrh metodik* 1. vyd Praha: Grada, 2005. ISBN 80-247-1075-7

[12] VONDRÁK, I. *Software Process* [online].
URL <http://vondrak.cs.vsb.cz/download.html/> [citováno 20. února 2010]